Tampereen yliopisto
Tampere University

COMP.SE.100-EN ItSE

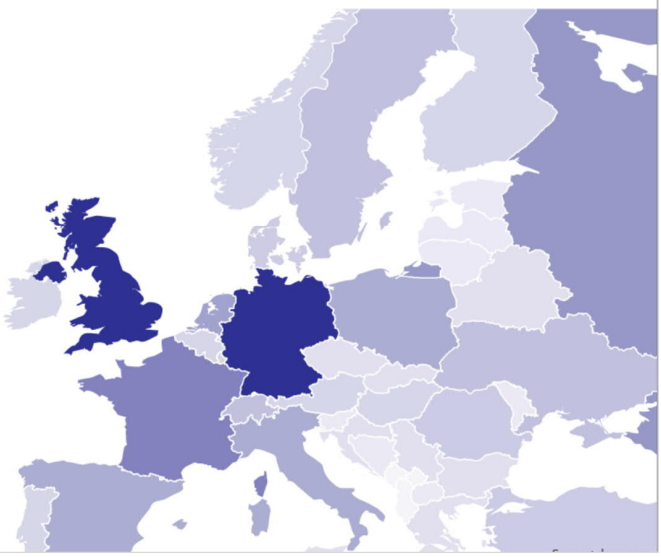
Zoom begins soon...

at 1415 o'clock.

THE NUMBER OF PROFESSIONAL SOFTWARE DEVELOPERS IN EUROPE BY COUNTRY

DAXX

Germany	873,398
United Kingdom	813,500
France	467,454
Russia	368,291
Netherlands	310,048
Italy	292,586
Spain	268,149
Poland	254,682
Ukraine	184,700
Sweden	175,794
Switzerland	144,382
Turkey	123,206
Belgium	108,626
Romania	105,170
Czech Republic	96,324
Denmark	95,391
Austria	92,772
Finland	82,874
Norway	79,112
Hungary	79,075



Tampereen yliopisto
Tampere University

COMP.SE.100-EN, 2020, course schedule v6c (02.09.2020)

week	lectures	exam	weekly exercises	project assignment (exercise work)	week
35	L1: course basics		--- sign to WE groups ---	sign for project = grouping...	35
36	Project Assignment explained		**WE1**: intro to requirements	grouping, groups to Moodle	36
37	L2: Sw Eng in general		**WE2**: Trello and agile way	group's Trello board ready with product backlog	37
38	L3: requirements		**WE3**: feasibility study and stakeholder analysis	working...	38
39	L4: basic UML diagrams		**WE4**: requirements	working...	39
40	L5: more UML diagrams	EXAM-1	**WE5**: UML diagrams - Use case	working...	40
41	L6: different sw systems	EXAM-1	**WE6**: UML diagrams - concept/entity and navigation	deadline for 1st phase documentation and presentation	41
42	examination week		examination week	examination week	42
43	L7: life cycle models		groups' 1st presentations	groups' 1st phase presentations	43
44	L8: quality and testing		**WE7**: development processes	feedback group-to-group at PRP, from 1st phase	44
45	L9: project work	Forms-2	**WE8**: testing and error reporting	deadline for diagrams first versions (Moodle)	45
46	L10: project management		**WE9**: effort estimation	feedback to groups from diagrams (from assistants)	46
47	L11: open source, APIs, IPR		**WE10**: delivery contracts and terms of use	deadline for 2nd phase presentation (PRP)	47
48	L12: embedded systems, IoT	EXAM-3	groups' final presentations	groups' final presentations / feedback g-to-g (PRP)	48
49	L13: recap, summary	EXAM-3	- - -	final (2.) delivery of project documentation	49
50	examination week		examination week	feedback inside group, student-to-student at PRP	50
51	examination week		examination week	end of game / game over.	51
	Lectures: Wed at 1415-16.		Weekly exercises:		
			Mon 0815-10	AUTUMN 2020 (1-2. periods)	
			Mon 1215-14	are remote/distant learning.	
			Tue 0815-10		
			Tue 1415-16		
			Wed 0815-10.		

Remote/distant learning 2020.
No contact teaching at ItSE 2020.

Tampereen yliopisto
Tampere University

COMP.SE.100 -EN "ItSE"

Introduction to Software Engineering

2020, 1-2. periods

5 credit units

07-lifecycle-ItSE-2020-v9

Tampereen yliopisto
Tampere University

COMP.SE.100-EN (ItSE)
Introduction to Software Engneering

Lecture 7, 21.10.2020

Tensu: remember to start Zoom lecture recording, at 1415

Prefer course Moodle over SISU information.

Students are recommended to follow Moodle News/messages.

DAXX

Company Developers Services Clients Blog Careers CONTACT US

How Many Software Developers Are in the US and the World?

POSTED FEB 09, 2020 TRENDS

How Many Software Developers Are There in the World?

According to **Evans Data Corporation**, there were 26,4 million software developers in the world in 2019, a number that in 2023 is expected to grow to 27,7 million and 28.7 million in 2024. The USA is taking the leading position by the number of software developers reached 4,2 million.

According to **IDC calculations**, in 2018 the number of software developers in the world grew to 22,3 million, while in 2014 there were only 18,5 million programmers.

Slashdata presented their statistics telling there were 18.9 million software developers in the world in 2019 and this number is going to reach 45 million in 2030.

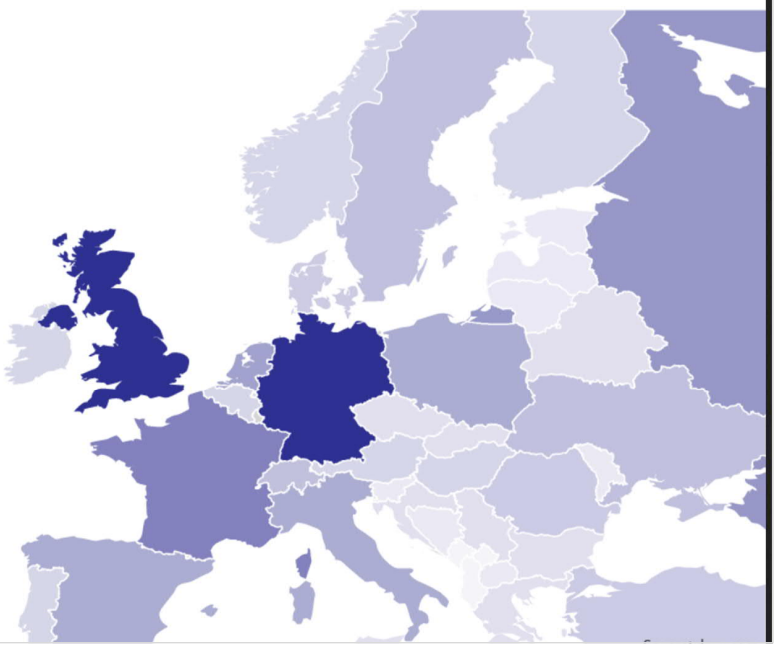
Year	Number of Software Developers
2018	23.9 million

Hire Developers in Ukraine

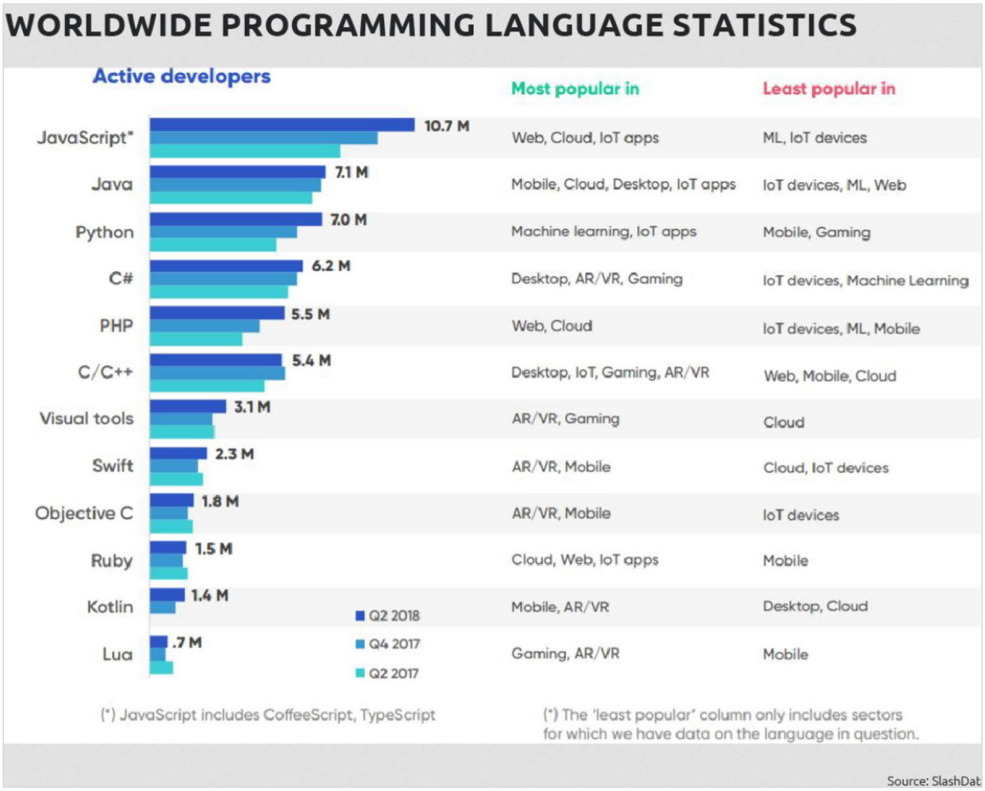
THE NUMBER OF PROFESSIONAL SOFTWARE DEVELOPERS IN EUROPE BY COUNTRY

DAXX

Germany	873,398
United Kingdom	813,500
France	467,454
Russia	368,291
Netherlands	310,048
Italy	292,586
Spain	268,149
Poland	254,682
Ukraine	184,700
Sweden	175,794
Switzerland	144,382
Turkey	123,206
Belgium	108,626
Romania	105,170
Czech Republic	96,324
Denmark	95,391
Austria	92,772
Finland	82,874
Norway	79,112
Hungary	79,075



Tampereen yliopisto
Tampere University



[https://www.daxx.com/blog/development-trends/number-software-developers-world]

Tampereen yliopisto
Tampere University

COMP.SE.100-EN, 2020, course schedule v6c (02.09.2020)

week	lectures	exam	weekly exercises	project assignment (exercise work)	week
35	L1: course basics		--- sign to WE groups ---	sign for project = grouping...	35
36	Project Assignment explained		**WE1**: intro to requirements	grouping, groups to Moodle	36
37	L2: Sw Eng in general		**WE2**: Trello and agile way	group's Trello board ready with product backlog	37
38	L3: requirements		**WE3**: feasibility study and stakeholder analysis	working...	38
39	L4: basic UML diagrams		**WE4**: requirements	working...	39
40	L5: more UML diagrams	EXAM-1	**WE5**: UML diagrams - Use case	working...	40
41	L6: different sw systems	EXAM-1	**WE6**: UML diagrams - concept/entity and navigation	deadline for 1st phase documentation and presentation	41
42	examination week		examination week	examination week	42
43	L7: life cycle models		groups' 1st presentations	groups' 1st phase presentations	43
44	L8: quality and testing		**WE7**: development processes	feedback group-to-group at PRP, from 1st phase	44
45	L9: project work	Forms-2	**WE8**: testing and error reporting	deadline for diagrams first versions (Moodle)	45
46	L10: project management		**WE9**: effort estimation	feedback to groups from diagrams (from assistants)	46
47	L11: open source, APIs, IPR		**WE10**: delivery contracts and terms of use	deadline for 2nd phase presentation (PRP)	47
48	L12: embedded systems, IoT	EXAM-3	groups' final presentations	groups' final presentations / feedback g-to-g (PRP)	48
49	L13: recap, summary	EXAM-3	- - -	final (2.) delivery of project documentation	49
50	examination week		examination week	feedback inside group, student-to-student at PRP	50
51	examination week		examination week	`end of game / game over.`	51
	Lectures: Wed at 1415-16.		Weekly exercises:		
			Mon 0815-10	AUTUMN 2020 (1-2. periods)	
			Mon 1215-14	are remote/distant learning.	
			Tue 0815-10		
			Tue 1415-16		
			Wed 0815-10.		

Tampereen yliopisto
Tampere University

Course contents (plan)

1. Course basics, intro
2. Sw Eng in general, overview
3. Requirements
4. Different software systems
5. Basic UML Diagrams (”Class”, Use Case, Navigation)
6. UML diagrams, in more detail
7. **Life Cycle models**
8. Quality and Testing
9. Project work
10. Project management
11. Open source, APIs, IPR
12. Embedded systems
13. Recap

Tampereen yliopisto
Tampere University

7. Life Cycle models (etc.)

- software life-cycle
- software development life-cycle (SDLC)
- different models
  - waterfall
  - spiral
  - iterative
  - incremental
  - agile (e.g. Scrum) and Scrum-BUT Scrum in more detail
  - XP (eXtreme Programming)
  - lean , MVP
  - DevOps
  - SAFe
  - kanban
  - PRINCE2
  - hybrid models...

Tampereen yliopisto
Tampere University

Current at course (week 43/2020)

- EXAM 2/3 (diagrams) is transferred to O365 Forms examination, due to worsening corona situation. It may perhaps be that EXAM classes could be closed down. (Moodle examinations are not recommended for large student groups.) So we move to Forms exam as pre-emptive action. The Forms (2nd) exam will be at week 45, on Wednesday 04.11.2020 starting at 1615 o'clock, after lecture time. Exam is about diagrams, the three same as you have in your project assignment (exercise work); context, use case, navigation.

- no WEs this week (1st presentations, week 43)
- only three WE groups continue at 2nd study period
- continue updating your Trello (kanban) boards = use at your process.

Tampereen yliopisto
Tampere University

First, general course matters

Juanita: groups G01-G04

Aleksius: ODD groups; G05,G07,G09,G11,G13,G15,G17,G19,G21,G23,G27

Lauri: EVEN groups; G06,G08,G10,G12,G14,G16,G18,G20,G22,G24,G28

- Trello board is used as help for work division and assignment

WE attendees:

- Mon 0815-10 9, 8,10, 5, 6, 4,
- **Mon 1215-14 11,12,12,13,11,12,**
- **Tue 0815-10 3, 6, 4, 6, 5, 5,**
- **Tue 1415-16 8,10, 9, 8, 5, 7,**
- Wed 0815-10 12,11, 9, 8, 7, 6,

Very small WEs are not reasonable, we discontinued two WE groups at 2nd study period.

Tampereen yliopisto
Tampere University

Now, the main contents today (L7) is methods, methodologies, processes and technologies...

most emphasis on agile and Scrum.

Scrum is the most popular agile methodology currently, although tens of other methods and methodologies exist.

Scrum fits best to small projects where new systems are developed, by experienced development teams. But you may also fail with Scrum if you do not know what you do or how.

Make you own methodology by adapting the ”best suitable practices” (methods) to your process !!

Methodology = a collection of methods.

Tampereen yliopisto
Tampere University

Software development life cycle

A FEW SLIDES AS REPLAY FROM PREVIOUS LETURES

SLC = software life cycle

software life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases may overlap or be performed iteratively. *Contrast with:* **software development cycle.**

In every Sw Eng project there is some
- preliminary analysis
- requirements specification
- design
- implementation
- testing
- documentation.

But methods and processes, way of working, have changed.

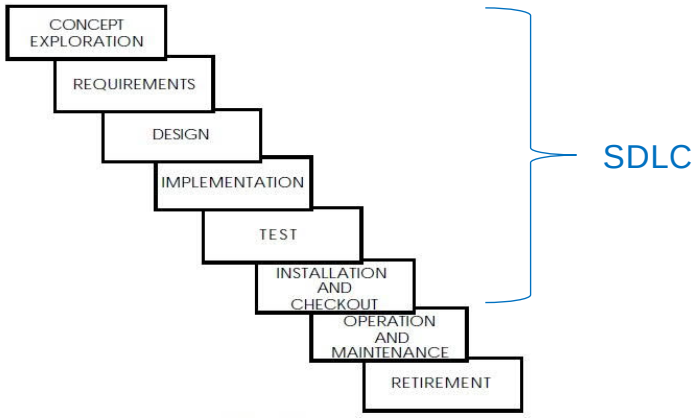


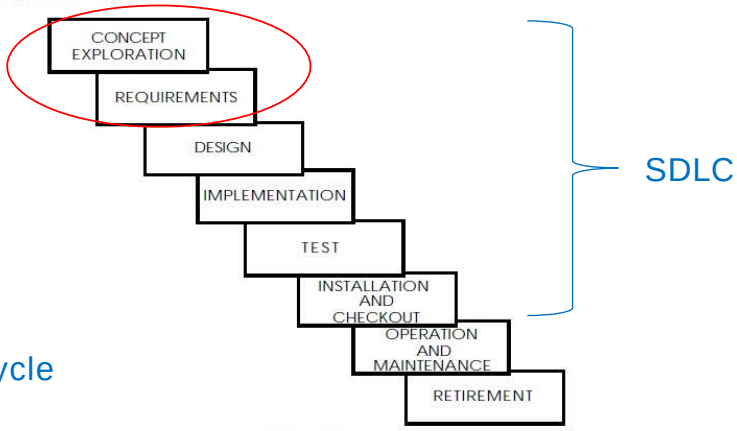
Fig 15
Sample Software Life Cycle

[IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990(R2002)]

SLC = software life cycle

software life cycle. The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. *Note:* These phases may overlap or be performed iteratively. *Contrast with:* **software development cycle.**

COMP.SE.100 ”ItSE” is about that small part



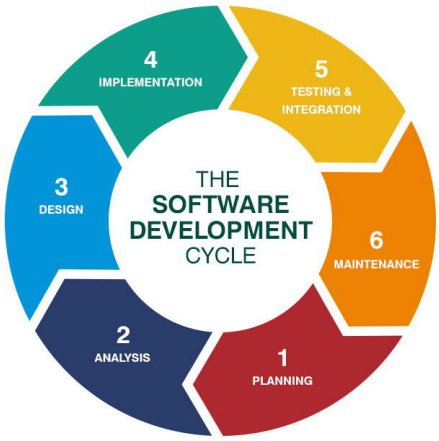
SDLC = software development life cycle

Fig 15
Sample Software Life Cycle

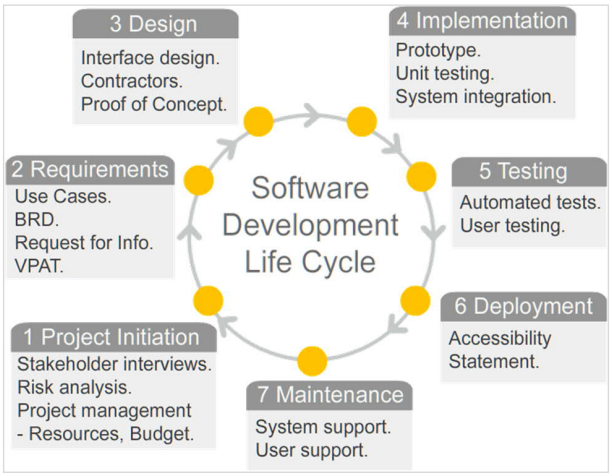
[IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990(R2002)]

“Waterfall” shaped as a SDLC circle

Well, in every Sw Eng project there is some
- preliminary analysis
- requirements specification
- design
- implementation
- testing
- documentation.



[Husson University]



[University Of Melbourne]

In every life cycle model there are the same basic items in some role.

BRD= business requirements document
VPAT= voluntary product accessibility template.

Tampereen yliopisto
Tampere University

Software life cycle

ISO/IEC/IEEE 24765:2017(E)
Systems and software engineering — Vocabulary
Second edition, 2017

3.3803
software development cycle

SDC = SDLC

1. period of time that begins with the decision to develop a software product and ends when the software is delivered
cf. software life cycle

3.3823
software life cycle (SLC)

SLC

1. project-specific sequence of activities that is created by mapping the activities of a standard onto a selected software life cycle model (SLCM) [IEEE 730-2014 IEEE Standard for Software Quality Assurance Processes, 3.2]
2. software system or software product cycle initiated by a user need or a perceived customer need and terminated by discontinued use of the product or when the software is no longer available for use.

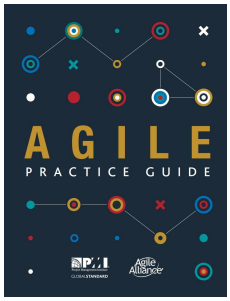


Table 3-1. Characteristics of Four Categories of Life Cycles

Characteristics				
Approach	**Requirements**	**Activities**	**Delivery**	**Goal**
Predictive	Fixed	Performed once for the entire project	Single delivery	Manage cost
Iterative	Dynamic	Repeated until correct	Single delivery	Correctness of solution
Incremental	Dynamic	Performed once for a given increment	Frequent smaller deliveries	Speed
Agile	Dynamic	Repeated until correct	Frequent small deliveries	Customer value via frequent deliveries and feedback

Tampereen yliopisto
Tampere University

Waterfall

Tampereen yliopisto
Tampere University

waterfall model (Royce, 1970)

Waterfall (FI: vesiputousmalli)

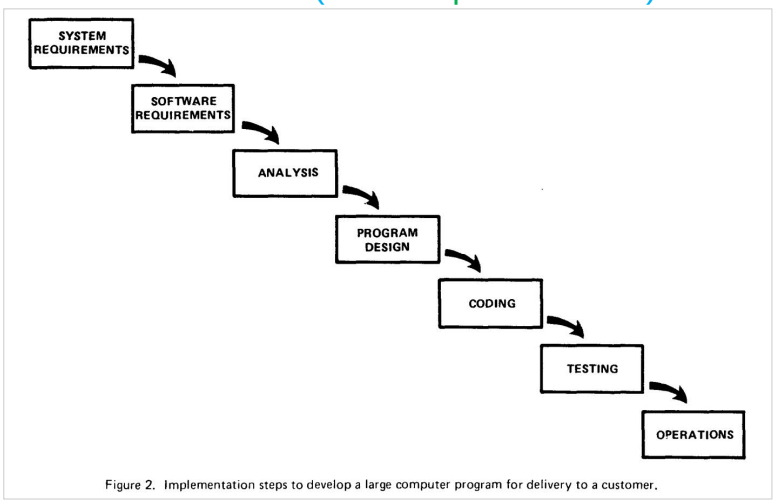


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

”Two-way waterfall” (FI: lohiputousmalli)

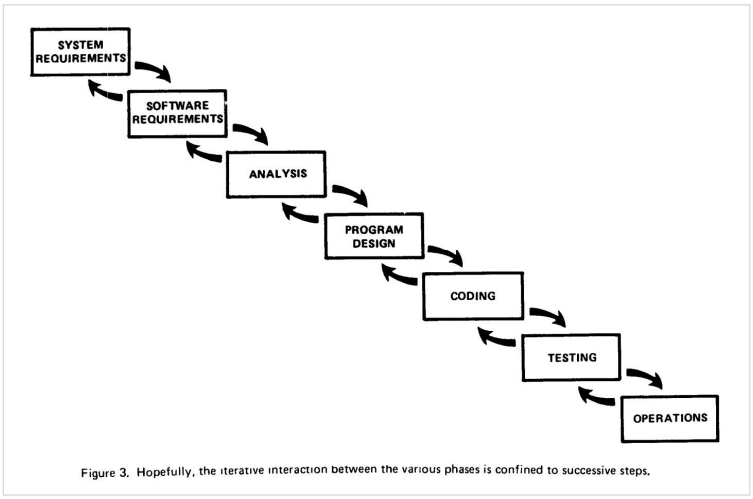


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

[Winston Royce: Managing the Development of Large Software Systems, 1970]

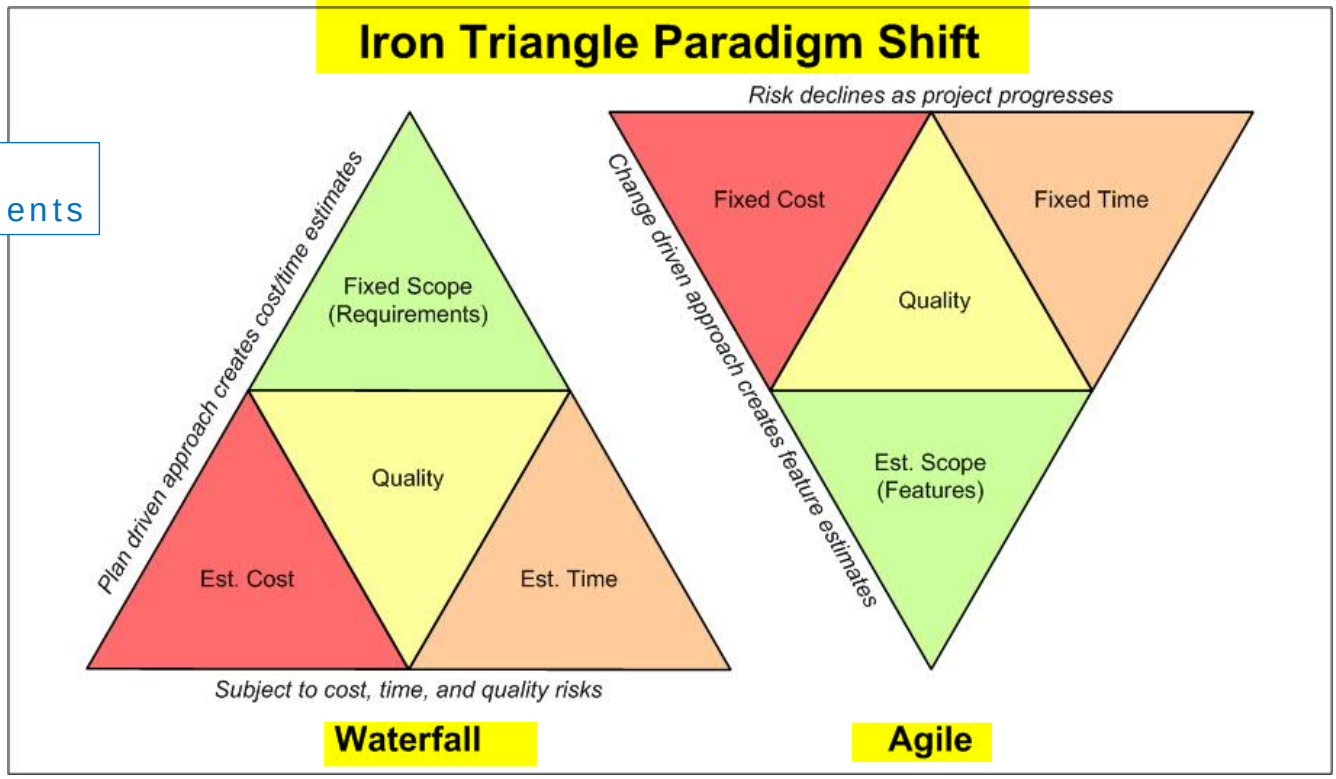
Tampereen yliopisto
Tampere University

Waterfall still suits to some projects

When to use the Waterfall Model

- requirements are very well known
- product definition is stable
- technology is understood
- new version of an existing product
- porting an existing product to a new platform.

Agile enhancements



[http://iq3group.blogspot.com/2013/01/decision-analysis-dcf-is-waterfall-rom.html]

Customer gets versions (phase releases) often

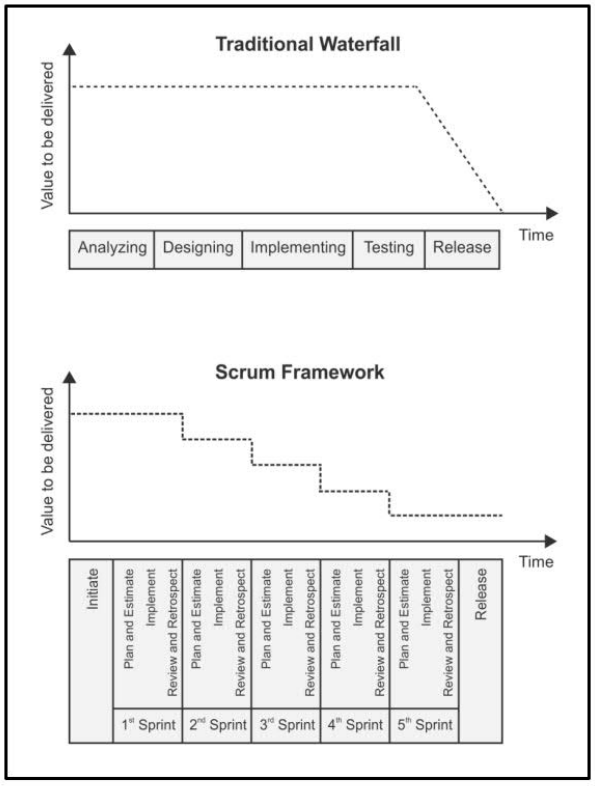
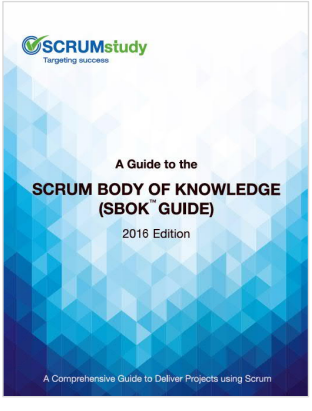
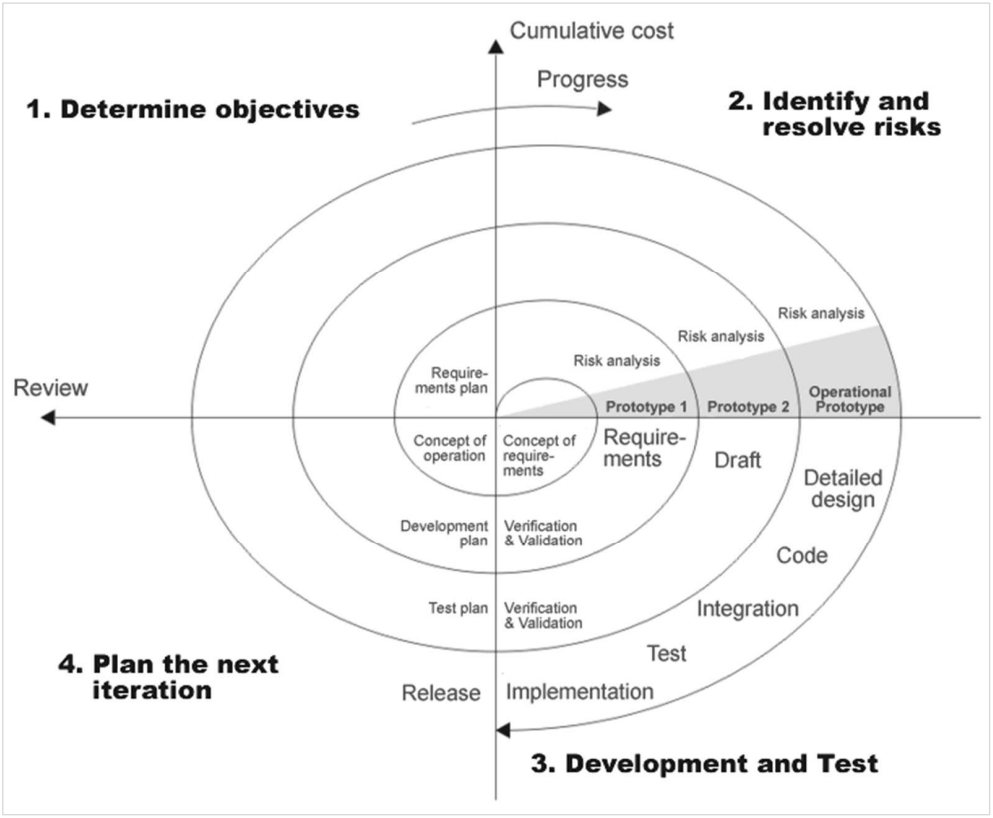


Figure 2-9: Scrum vs. Traditional Waterfall



Spiral

Boehm's Spiral model (1986)



Tampereen yliopisto
Tampere University

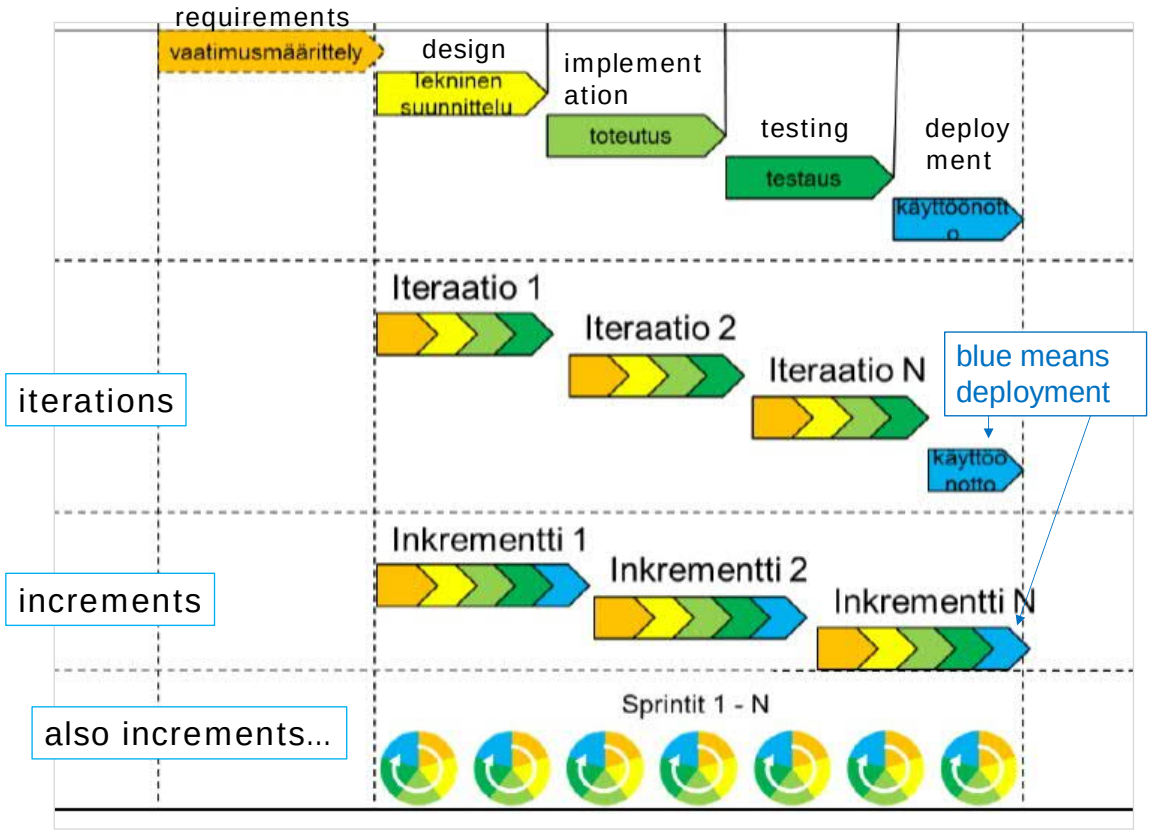
Iterative Incremental

Tampereen yliopisto
Tampere University

iterative vs. incremental

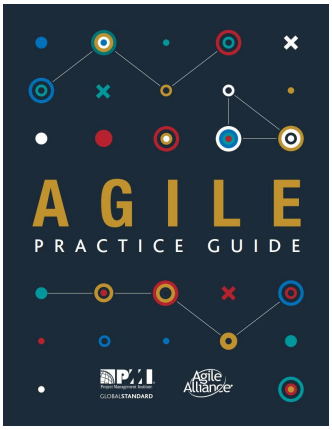
At iterative way you make iterations, and at the end you publish (deploy) the product.

At incremental way you make iteration, and deploy a product version after every increment.



[picture source: Apotti files, in Finnish]

Agile



And in street talk ”agile” may mean almost what ever non-traditional sw dev...

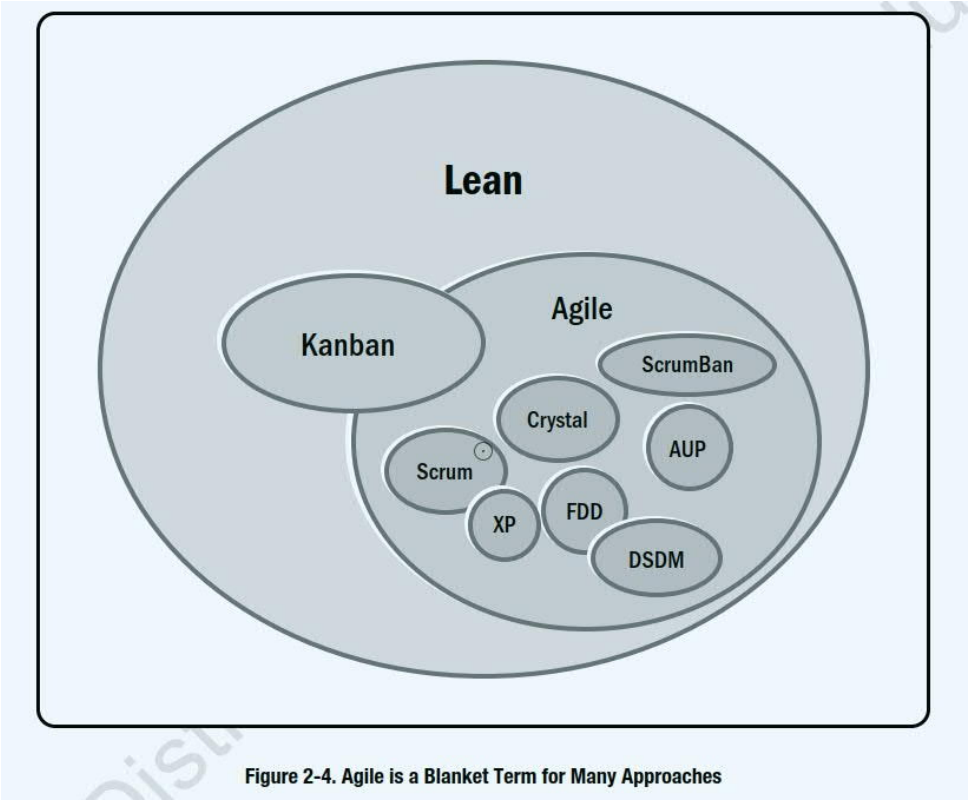


Figure 2-4. Agile is a Blanket Term for Many Approaches

Agile Manifesto

The core is there

But still the small text is also needed, although it is less important.

[https://agilemanifesto.org/]

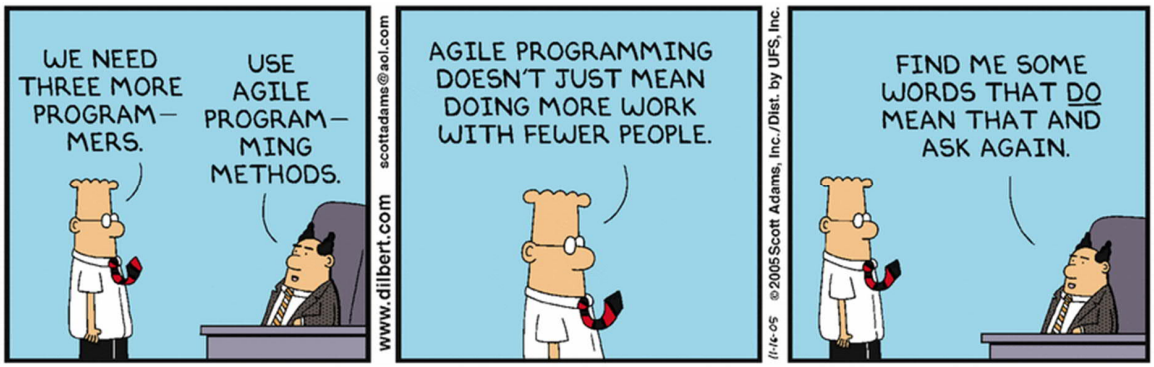
Principles behind the Agile Manifesto

We follow these (12) principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity -- the art of maximizing the amount of work not done -- is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

[https://agilemanifesto.org/principles.html]

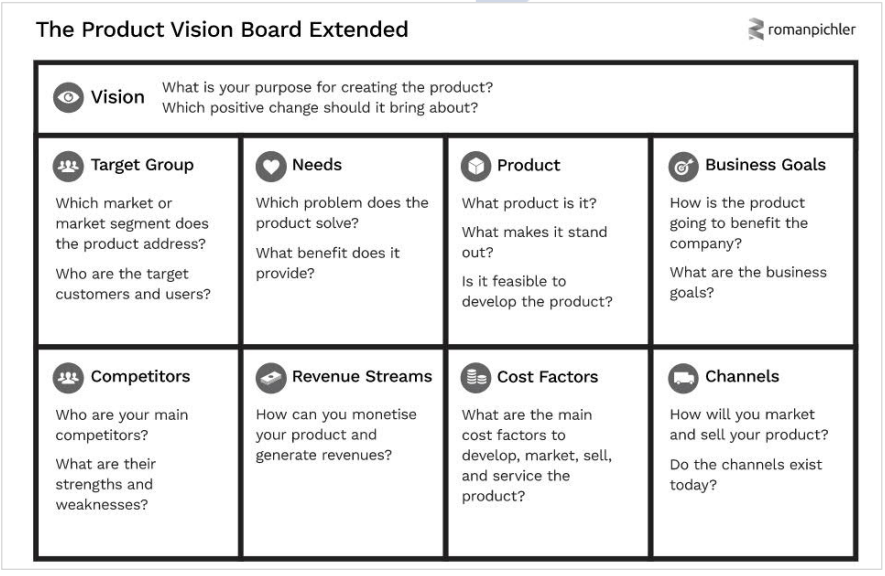
Well, not just that…



Start from product vision

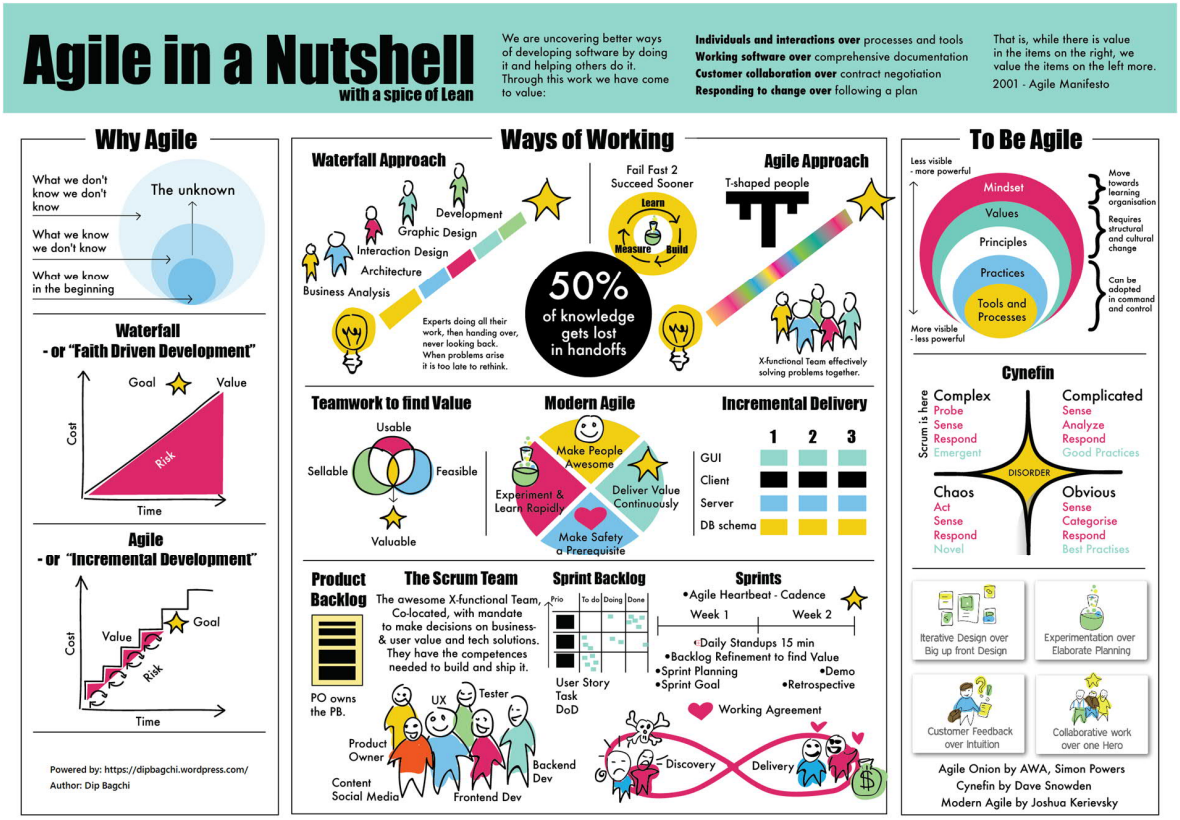
(think before act)

[https://worldofagile.com/blog/product-strategy-and-product-roadmap/]

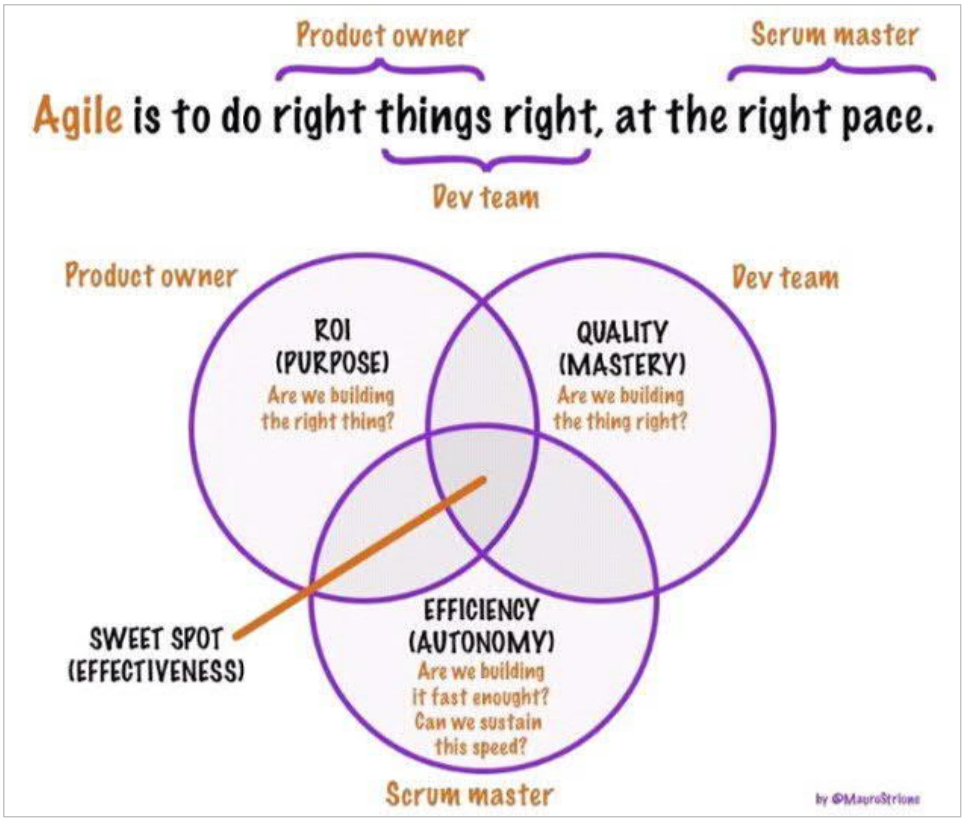


Tampereen yliopisto
Tampere University

Some good details here

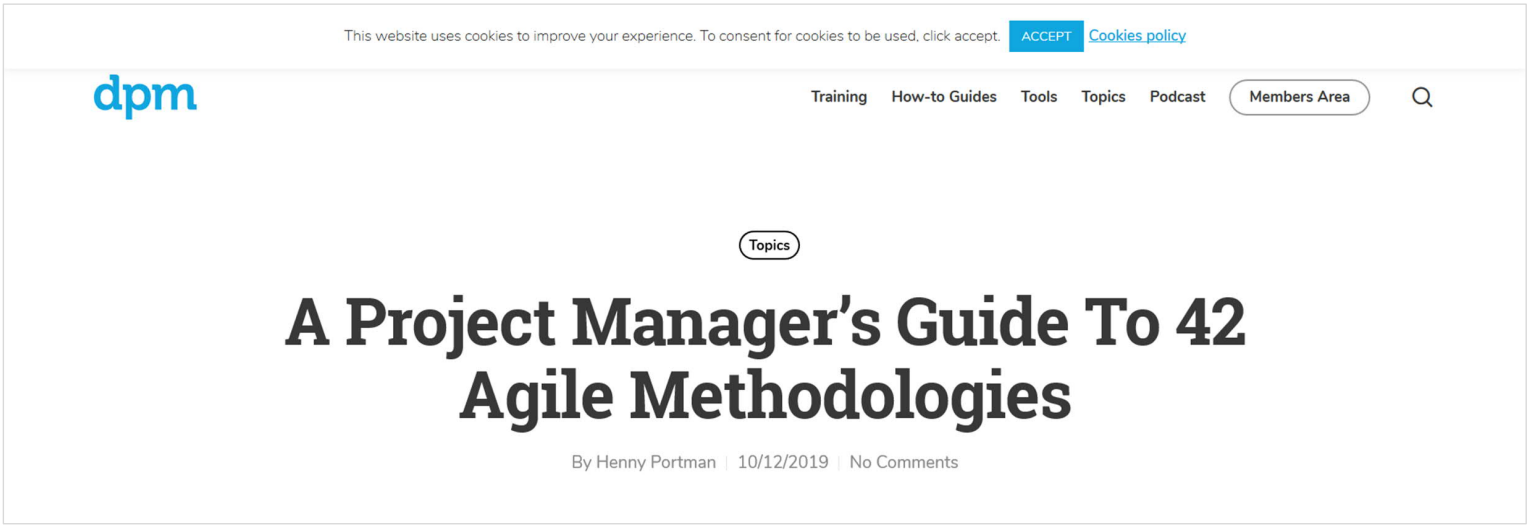


Tampereen yliopisto
Tampere University



Tampereen yliopisto
Tampere University

https://thedigitalprojectmanager.com/agile-methodologies/



Tampereen yliopisto
Tampere University

agile, videos and glossary

At YouTube there are many videos e.g. by Mountain Goat Software (www.mountaingoatsoftware.com/) or Mike Cohn.

Agile Estimating and Planning: Planning Poker - Mike Cohn
https://www.youtube.com/watch?v=MrIZMuvjTws

You may also check:
https://www.agilealliance.org/agile101/agile-glossary/

Agile vs. Scrum, methodologies

Agile is a continuous iteration of development and testing in the software development process whereas Scrum is an Agile process to focus on delivering the business value in the shortest time. (well, not always strictly that way…)

In the Agile process, leadership plays a vital role; on the other hand, Scrum fosters a self-organizing, cross-functional team.

Agile involves collaborations and face-to-face interactions between the members of various cross-functional teams whereas Scrum collaboration is achieved in daily stand up meetings. (well, not always such…)

In Agile process design and execution should be kept simple whereas in Scrum process design and execution can be innovative and experimental.

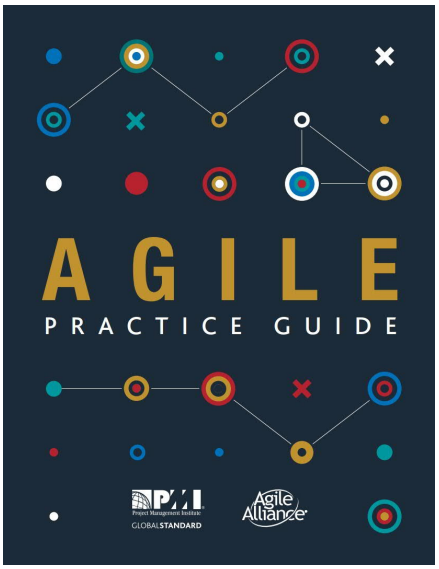
Agile is a development methodology based on iterative and incremental approach.

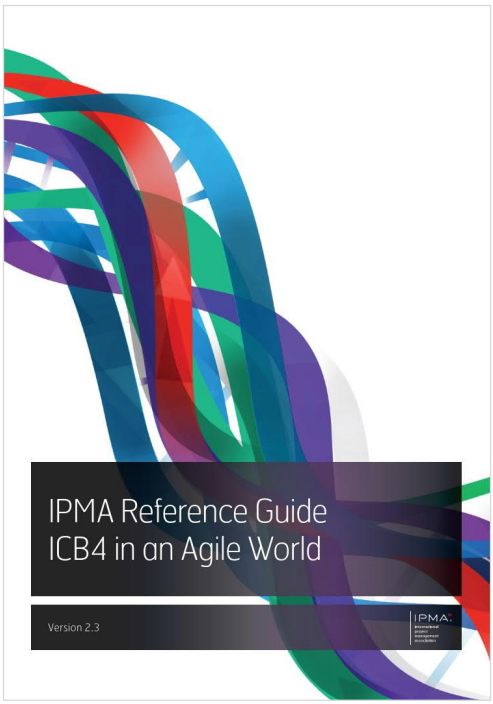
Scrum is one of the implementations of agile methodology. In which incremental builds are delivered to the customer in every two to three weeks' time.

[https://www.guru99.com/agile-vs-scrum.html]

Methodology is a collection of methods.

state-of-agile-2020.pdf

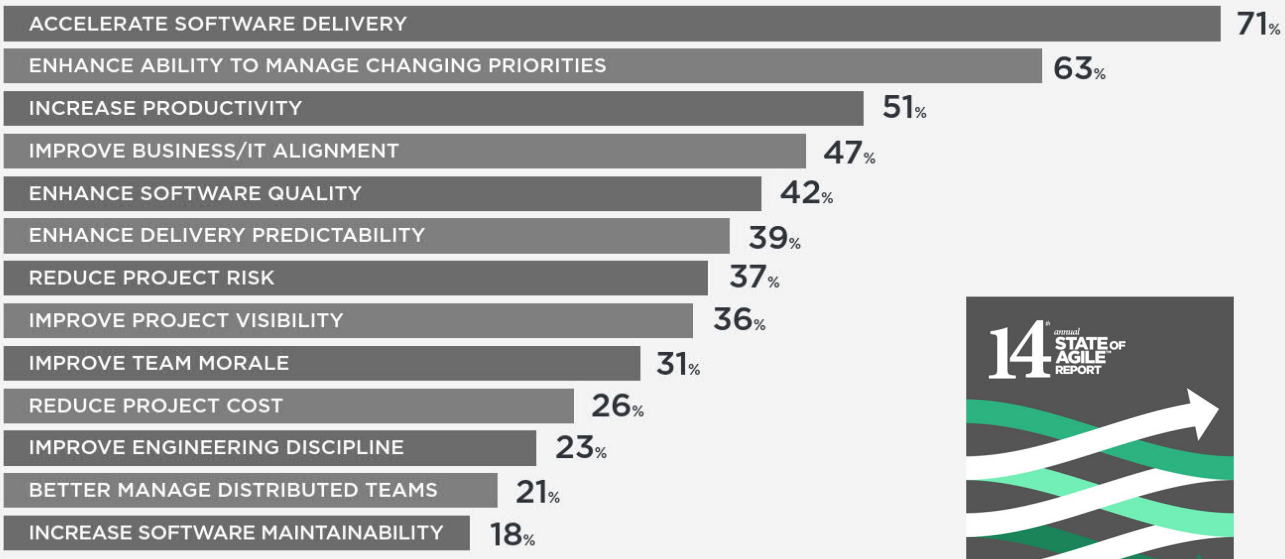




Tampereen yliopisto
Tampere University

REASONS FOR ADOPTING AGILE

Accelerating software delivery and enhancing ability to manage changing priorities remain the top reasons stated for adopting Agile. Respondents indicated this year that reasons for adoption were less about reducing project cost (26% compared to 41% last year), and more about reducing project risk (37% compared to 28% last year).

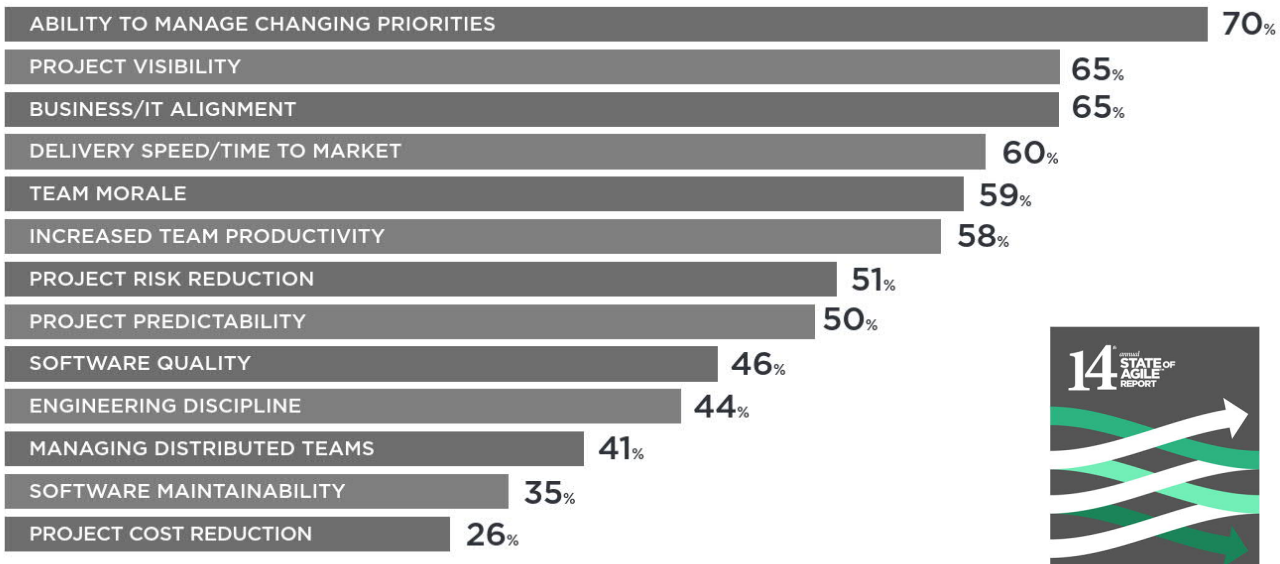


*Respondents were able to make multiple selections

Tampereen yliopisto
Tampere University

BENEFITS OF ADOPTING AGILE

We continue to see many benefits realized by companies adopting Agile. The theme of the top 5 reported benefits is speed and adaptability. This corresponds with the top reported reasons for adopting Agile.

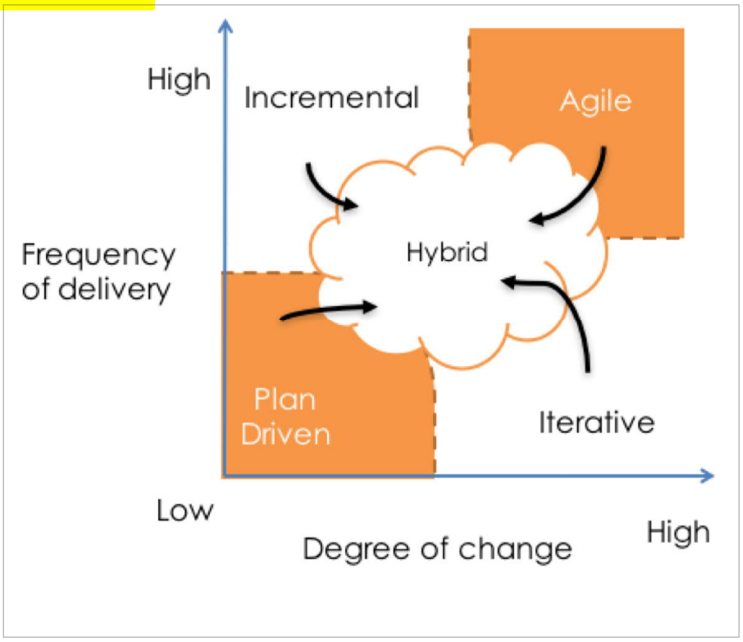


*Respondents were able to make multiple selections

hybrid agile models

Blended Agile is the combination of two or more established Agile methods, techniques, or frameworks.

Hybrid Agile is the combination of Agile methods with other non-Agile techniques.



[https://www.agilealliance.org/what-is-hybrid-agile-anyway/]

Tampereen yliopisto
Tampere University

Planning poker (estimation poker)

Tampereen yliopisto
Tampere University

Planning poker

Planning Poker is an agile estimating and planning technique that is consensus based. To start a poker planning session, the product owner or customer reads an agile user story or describes a feature to the estimators.

The estimators discuss the feature, asking questions of the product owner as needed. When the feature has been fully discussed, each estimator privately selects one card to represent his or her estimate. All cards are then revealed at the same time.

If all estimators selected the same value, that becomes the estimate. If not, the estimators discuss their estimates. The high and low estimators should especially share their reasons. After further discussion, each estimator reselects an estimate card, and all cards are again revealed at the same time.

The poker planning process is repeated until consensus is achieved or until the estimators decide that agile estimating and planning of a particular item needs to be deferred until additional information can be acquired.

[https://www.mountaingoatsoftware.com/agile/planning-poker]

Tampereen yliopisto
Tampere University

planning poker

A playful approach to estimation, used by many Agile teams.

The team meets in presence of the customer or Product Owner. Around the table, each team member holds a set of playing cards, bearing numerical values appropriate for points estimation of a user story.

The Product Owner briefly states the intent and value of a story. Each member of the development team silently picks an estimate and readies the corresponding card, face down. When everyone has taken their pick, the cards are turned face up and the estimates are read aloud.

The two (or more) team members who gave the high and low estimate justify their reasoning. After brief discussion, the team may seek convergence toward a consensus estimate by playing one or more further rounds.

(sometimes you may use this “wrong” and estimate working hours, or days at start)

A good Planning Poker video at YouTube

https://www.youtube.com/watch?v=MrIZMuvjTws

And there are also many more...

Pair programming

Tampereen yliopisto
Tampere University

Pair programming (also documenting etc.)

Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator"; it is expected that the programmers swap roles every few minutes or so.

More simply "pairing"; the phrases "paired programming" and "programming in pairs" are also used, less frequently. In general it is just working in pairs, together.

Common Pitfalls

- both programmers must be actively engaging with the task throughout a paired session, otherwise no benefit can be expected
- a simplistic but often raised objection is that pairing "doubles costs"; that is a misconception based on equating programming with typing – however, one should be aware that this is the worst-case outcome of poorly applied pairing
- at least the driver, and possibly both programmers, are expected to keep up a running commentary; pair programming is also "programming out loud" – if the driver is silent, the navigator should intervene
- pair programming cannot be fruitfully forced upon people, especially if relationship issues, including the most mundane (such as personal hygiene), are getting in the way; solve these first!

[www.agilealliance.org/glossary/]

Tampereen yliopisto
Tampere University

Refactoring

Tampereen yliopisto
Tampere University

refactoring code, ”fine-tuning”

Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior. For example, your “brute force” code is tested and works, but you re-write it to be a short elegant and well-commented code.

The following are claimed benefits of refactoring:

• refactoring improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance

• refactoring helps code understanding

• refactoring encourages each developer to think about and understand design decisions, in particular in the context of collective ownership / collective code ownership

• refactoring favors the emergence of reusable design elements (such as design patterns) and code modules.

Use e.g. SonarQube tool to find out code quality.

Tampereen yliopisto
Tampere University

Refactoring code

Refactoring is a tool specific to software projects. The aim of this technique is to improve the maintainability of the existing code and make it simpler, more concise, and more flexible.

Refactoring means improving the design of the present code without changing how the code behaves.

It involves the following:

- Eliminating repetitive and redundant code
- Breaking methods and functions into smaller routines
- Clearly defining variables and method names
- Simplifying the code design
- Making the code easier to understand and modify.

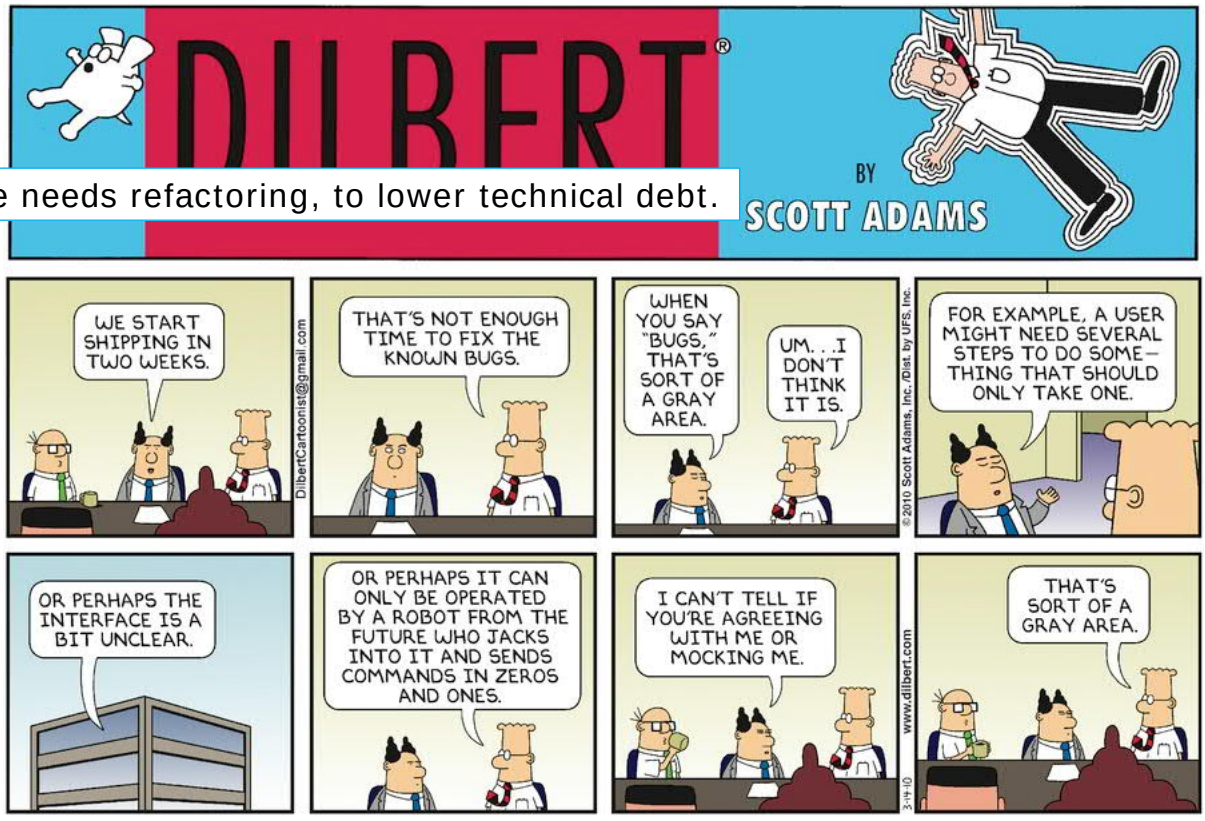
[SBOK Guide 2016]

Some refactoring types

Move and Rename Class
Extract Class
Extract Subclass
Extract Variable
Inline Variable
Parameterize Variable
Rename Variable
Rename Parameter
Rename Attribute
Move and Rename Attribute
Replace Variable with Attribute
Replace Attribute (with Attribute)
Merge Variable
Merge Parameter
Merge Attribute
Split Variable
Split Parameter
Split Attribute
Change Variable Type
Change Parameter Type
Change Return Type
Change Attribute Type
Extract Attribute
Move and Rename Method
Move and Inline Method

[https://github.com/lamchau/refactoring-exercise]

That code needs refactoring, to lower technical debt.



Tampereen yliopisto
Tampere University

Definition of Done (DoD)

Tampereen yliopisto
Tampere University

Done Criteria

Done Criteria are a set of rules that are applicable to all User Stories. A clear definition of Done is critical, because it removes ambiguity from requirements and helps the team adhere to mandatory quality norms.

This clear definition is used to create the Done criteria that are an output of the Create Prioritized Product Backlog process. A User Story is considered done when it is demonstrated to and approved by the Product Owner who judges it on the basis of the Done Criteria and the User Story Acceptance Criteria.

[SBOK Guide 2016]

DoD

- The definition of done is an agreed upon list of the activities deemed necessary to get a product increment, usually represented by a user story, to a done state by the end of a sprint.

[Agile Alliance]

"The most complex features are 20 % of requirements, and they take 80 % of the development time." (folklore)

Definition of... Done (Scrum Guide)

Definition of "Done"

- When a Product Backlog item or an Increment is described as "Done", everyone must understand what "Done" means. Although this varies significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of "Done" for the Scrum Team and is used to assess when work is complete on the product Increment.
- The same definition guides the Development Team in knowing how many Product Backlog items it can select during a Sprint Planning. The purpose of each Sprint is to deliver Increments of potentially releasable functionality that adhere to the Scrum Team's current definition of "Done."
- As Scrum Teams mature, it is expected that their definitions of "Done" will expand to include more stringent criteria for higher quality. Any one product or system should have a definition of "Done" that is a standard for any work done on it.

[Scrum Guide 2017]

DoD example; insurance company's new app enabling customers to report accidents. The app should also include a server-side backend.

- DoD will very likely evolve during the project, to be more complete.
- Product may be just accepted by the customer.
- Development team leaders check that DoD recommendations and requlations are met.

Project

- as backogs are in priority order; important features done and tested
- end-user system and UX tests done
- necessary documentation done and updated
- system / acceptance tests: functional, usability, performance, security, stress, compatibility
- accepted by customer
- functionality comply with backlog / user stories
- functional requirements implemented and tested
- also non-functional requirements verified
- possible restrictions (3rd class of requirements) checked
- code is maintainable (e.g. not spaghetti, good comments).

DoD example; insurance company's new app enabling customers to report accidents. The app should also include a server-side backend.

Sprint

- code (peer) reviews
- unit tests written
- unit tests passed (preferable automated, CI-pipeline)
- static code analysers, coverage metrics
- code comments written
- integration tests passed
- feature/functionality meets requirements
- feature acceptec by product owner / customer.

Other matters (at the end)

- at the end, product backlog usually has some (low priority) items
- refactoring not necessarily complete
- not all bugs need to be fixed.

Tampereen yliopisto
Tampere University

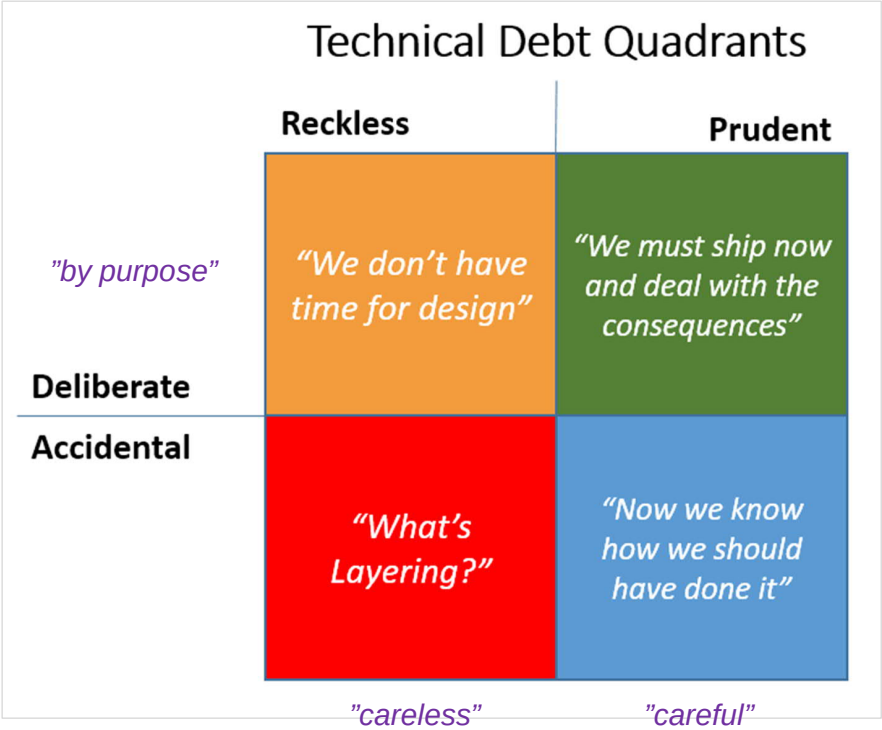
Technical debt

Tampereen yliopisto
Tampere University

technical debt

Technical debt is a metaphor for all of the shortcuts, hacks, and poor design choices made for a given software system that compromised its quality, usually in an attempt to meet a deadline.

It can be an appropriate business decision to take on technical debt, but if such debt is allowed to grow, the lack of quality of the system may eventually make it too expensive to maintain, at which point the business or team may need to declare “technical bankruptcy” and rewrite the application rather than continue to try to maintain it.



[https://deviq.com/technical-debt/]

Technical debt

Technical debt is an essential phenomenon that should be acknowledged by companies to understand its role in software development.

- The definition of technical debt is unclear, and it needs to be addressed more with research and discussion.
- Technical debt can be intentional, as a strategic decision to incur technical debt, and unintentional, as a negative side-effect of causing technical debt unintentionally.
- Technical debt is caused for both technical reasons in software and organizational reasons with people in software development.
- Technical debt can be also caused by legacy software over a long period of time.
- Taking technical debt can be beneficial in the short term with time-to-market and increased development speed, but having technical debt can be detrimental in the long term with omitted software quality and development productivity.
- All technical debt is not necessarily bad and can be sometimes left unpaid in software development.
- Technical debt management is a process including various technical activities done with a software and organizational activities done with social and political aspects in software development and economics.
- Technical debt management can be described with various maturity levels similar to the capability maturity model (CMM).

[Jesse Yli-Huumo: THE ROLE OF TECHNICAL DEBT IN SOFTWARE DEVELOPMENT, 2017]

Indicators of Technical Debt

The only one who can change this code is ”Ahto Simakuutio”.

Let us finish the testing in next release.

Let us just copy and paste this part.

It is ok for now (e.g. this sprint) but we will refactor it later.

I know if I touch that code everything else breaks.

Release (sprint deadline ☺) is coming up, so just get it done!

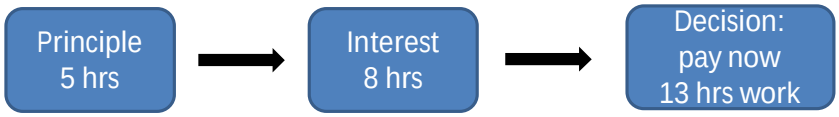
”TODO” in code, ”now hurry, will fix on Monday”.

Does anybody know where to change the player character shape (course assignment).

What should be done: update documentaion and comment the code.

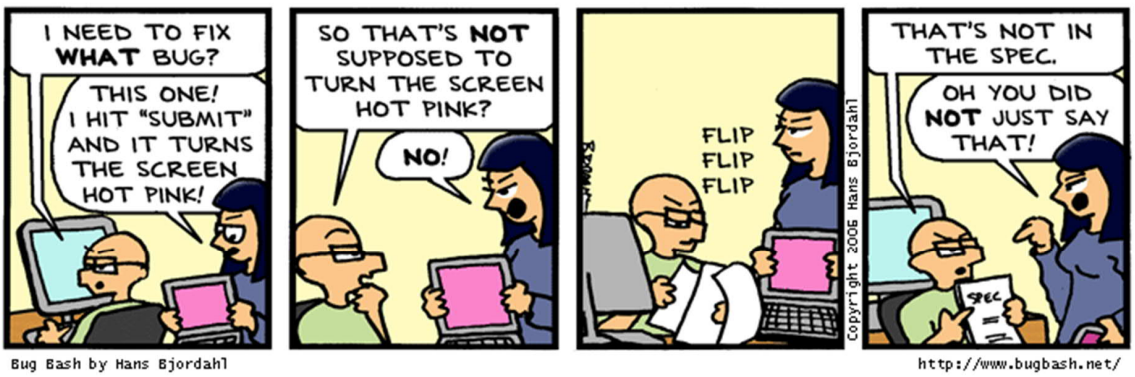
Paying debt: Example

- Debt item: Refactoring the artificial intelligence module in your course assignment
- Cost to refactor (i.e., principal) after Sprint 2 → 5 hrs
- During Sprint 3 you may have added new functionality to that module
- Cost to refactor it after Sprint 3 : 5 hrs (principal) + 8 hrs (interest) → 13 hrs
- In this case:



Tampereen yliopisto
Tampere University

TODO comments are anti-patterns



Extreme programming (XP)

XP = Extreme Programming

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

The general characteristics where XP is appropriate were described by Don Wells on www.extremeprogramming.org:

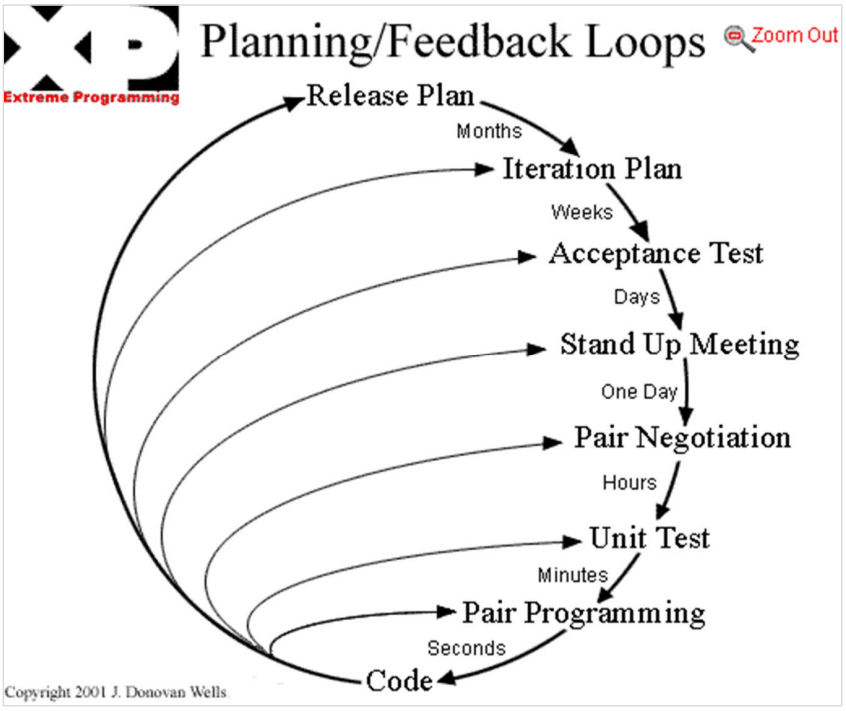
- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, co-located extended development team
- The technology you are using allows for automated unit and functional tests.

Due to XP's specificity when it comes to it's full set of software engineering practices, there are several situations where you may not want to fully practice XP. The post When is XP Not Appropriate on the C2 Wiki is probably a good place to start to find examples where you may not want to use XP.
While you can't use the entire XP framework in many situations, that shouldn't stop you from using as many of the practices as possible given your context.

[www.agilealliance.org/glossary/]

Tampereen yliopisto
Tampere University

XP loops



Note the quick feedback (reactions).

Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

COMP.SE.100-EN (ItSE)
Introduction to Software Engneering

Lecture 7, 21.10.2020

Tensu: remember to pause Zoom lecture recording

Zoom lecture break, 10 minutes stretching, walking, etc.

Tampereen yliopisto
Tampere University

Tampereen yliopisto
Tampere University

Scrum

Tampereen yliopisto
Tampere University

Scrum, shortly explained

Is that current ”silver bullet” hype or what ? Well, at least the most popular method at the moment... so, some hype also.

Experienced team concentrates to one project at a time. Team members know each other from many projects, and knows everybody's skills, strenghts and weaknesses. Team is self-guided (motivated).

Customer tells requirements, Product Owner (PO) brings those to development team (Product Backlog, PB), team decides what features/functionalities it takes (Sprint Backlog, SB), and splits those to work items (tasks).

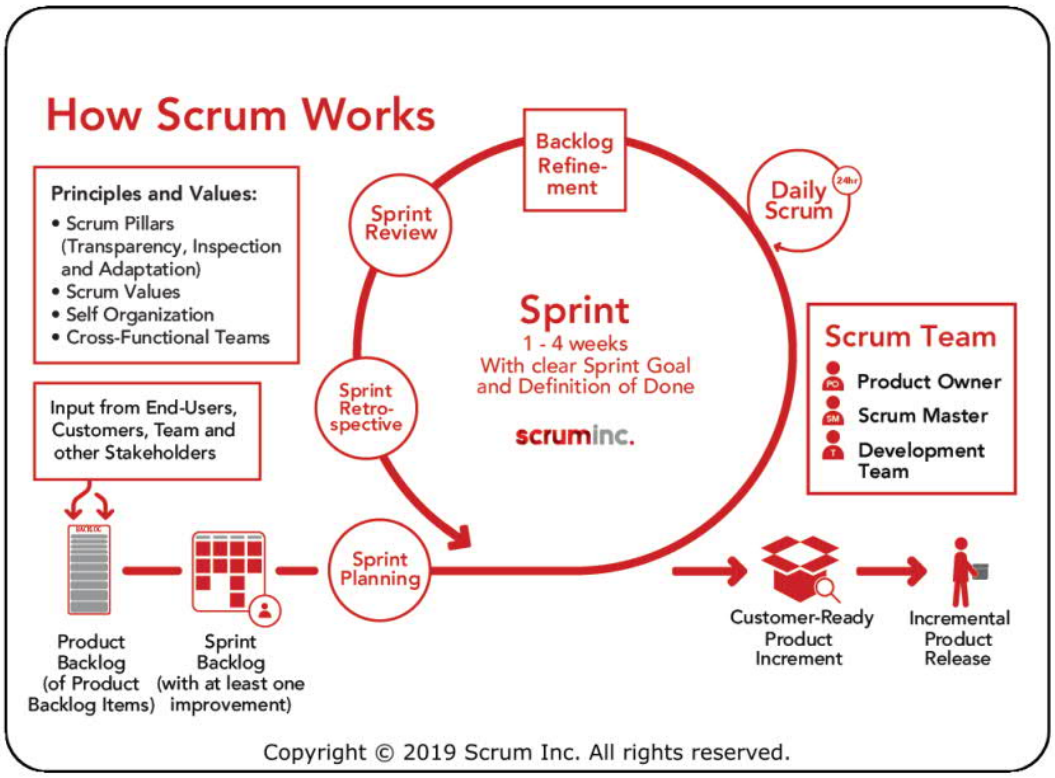
Work is done in iterations (Sprints), 2..4 weeks, after which demo (phase product) is shown to customer, for comments.

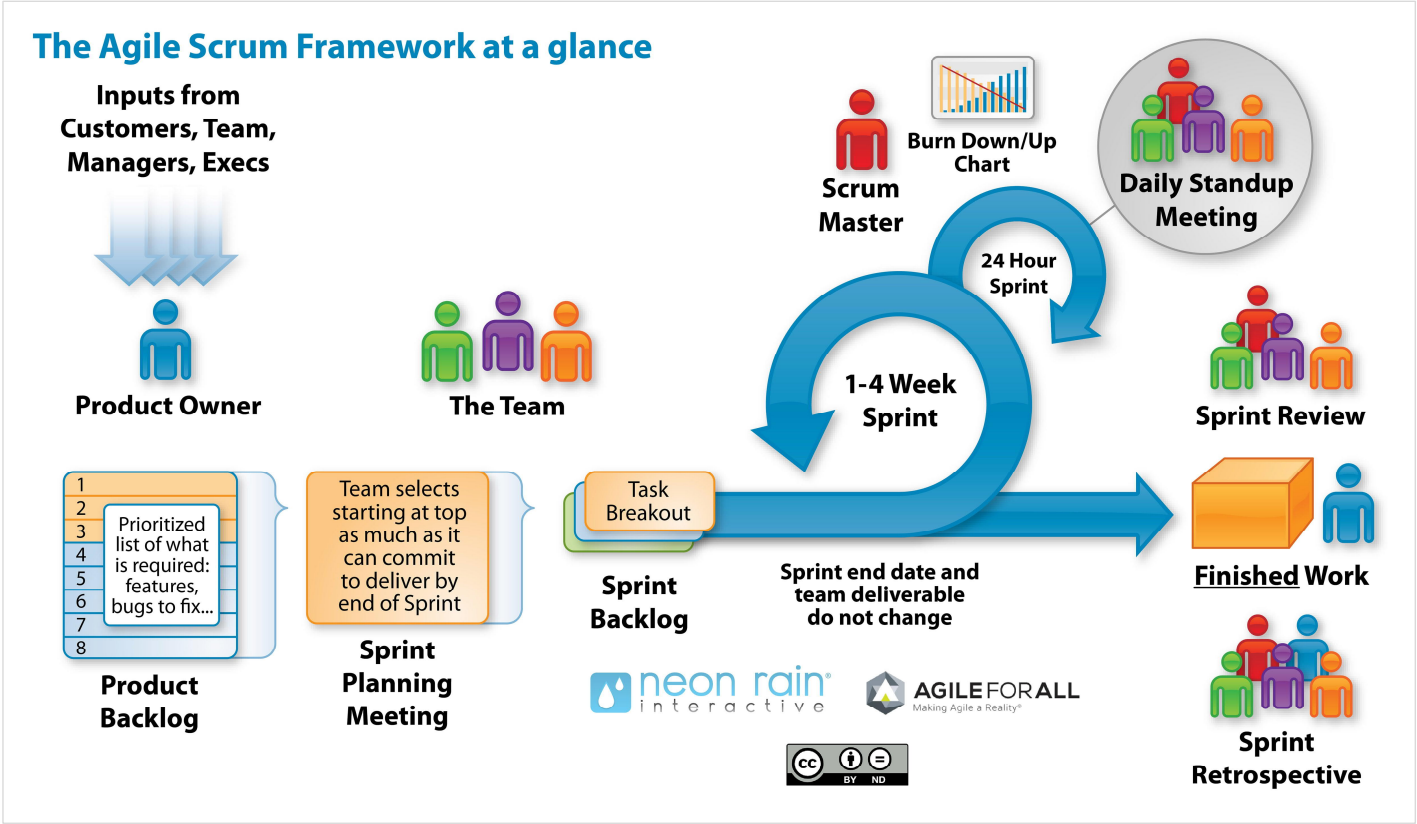
Daily meeting (stand-up) ensures situational awareness.

Documentation is at minimum.

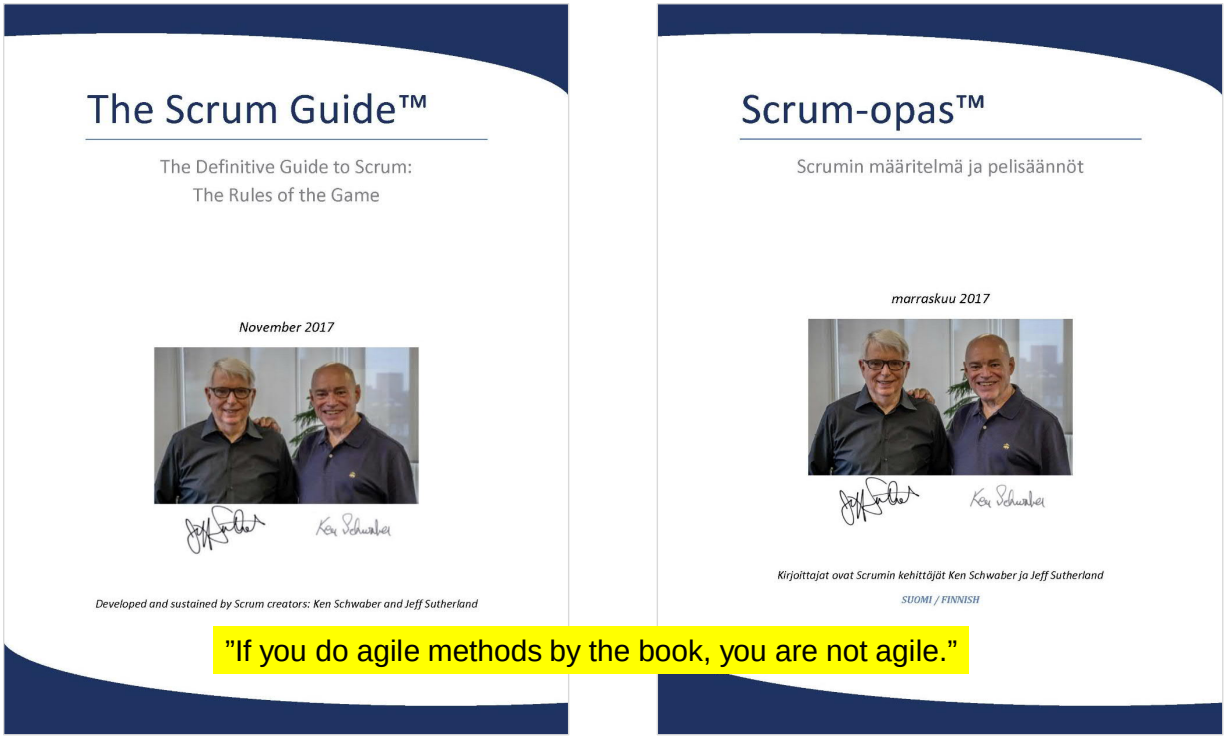
Remember: Scrum Master is not Project Manager.

Remember: overtime working is not an option.


How Scrum Works
Principles and Values:
• Scrum Pillars (Transparency, Inspection and Adaptation)
• Scrum Values
• Self Organization
• Cross-Functional Teams
Input from End-Users, Customers, Team and other Stakeholders
Backlog Refine-ment
Sprint Review
Daily Scrum
Sprint
1 - 4 weeks
With clear Sprint Goal and Definition of Done
scruminc.
Sprint Retro-spective
Sprint Planning
Scrum Team
Product Owner
Scrum Master
Development Team
Product Backlog (of Product Backlog Items)
Sprint Backlog (with at least one improvement)
Customer-Ready Product Increment
Incremental Product Release
Copyright © 2019 Scrum Inc. All rights reserved.


The Agile Scrum Framework at a glance
Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Burn Down/Up Chart
Daily Standup Meeting
24 Hour Sprint
1-4 Week Sprint
Sprint Review
Finished Work
Prioritized list of what is required: features, bugs to fix...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Sprint Retrospective
neon rain interactive
AGILE FOR ALL
Making Agile a Reality®

www.scrumguides.org



”If you do agile methods by the book, you are not agile.”

Tampereen yliopisto
Tampere University

Scrum values [Scrum Guide 2017]

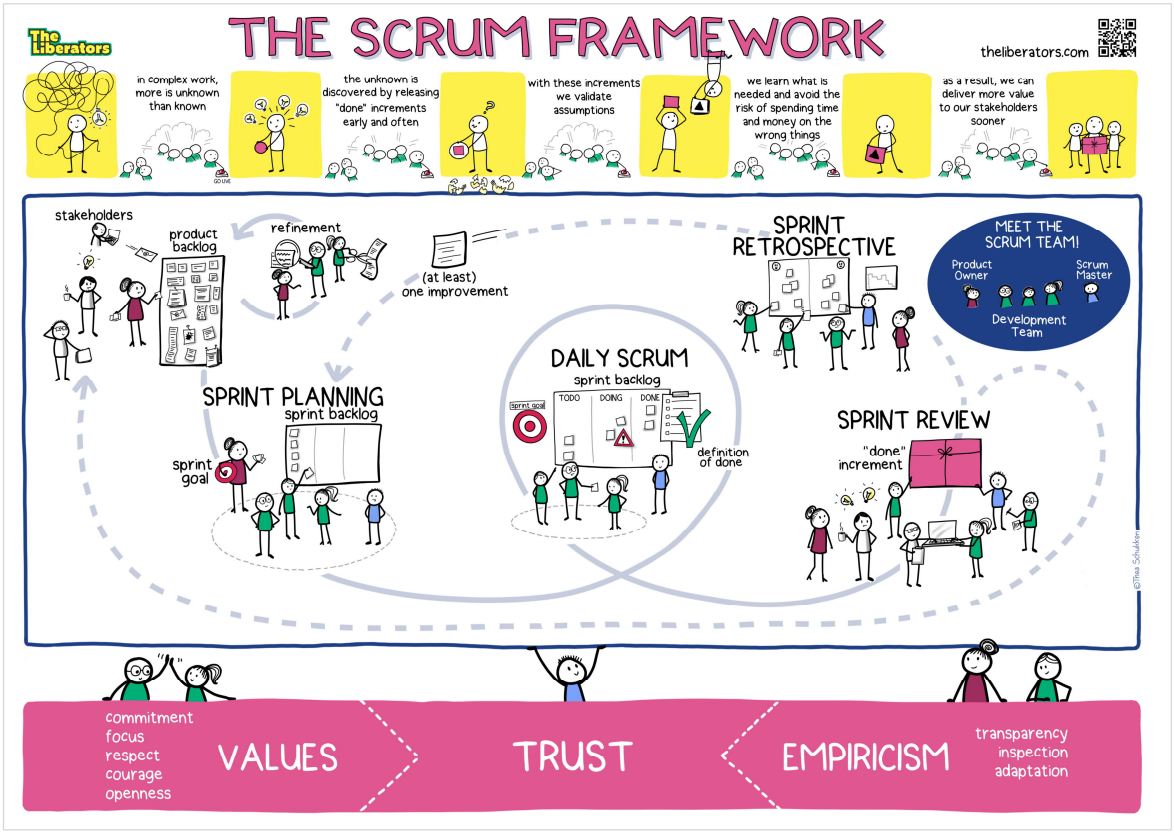
When the values of commitment, courage, focus, openness and respect are embodied and lived by the Scrum Team, the Scrum pillars of transparency, inspection, and adaptation come to life and build trust for everyone.

(FI: Scrumin arvot ovat sitoutuminen, rohkeus, keskittyminen, avoimuus ja kunnioitus. Arvojen mukaan toimiessaan Scrum-tiimi vahvistaa Scrumin kolmea tukijalkaa: läpinäkyvyyttä, tarkastelua ja sopeuttamista. Tämä puolestaan kasvattaa luottamusta.)

Tampereen yliopisto
Tampere University

Scrum terminology

Product Backlog (PB) (FI: tuotepino, tuotejono, kehitysjono, kehityspino)
Sprint (FI: pyrähdys), iteration, (some may call it increment (FI: tuoteversio))
Sprint Backlog (SB) (FI: tehtäväpino, tehtäväjono), work items to be done in the Sprint
Daily scrum (FI: päivittäispalaveri)
(PB) Work item, e.g. feature, user story (FI: kehitettävä asia / kehitysjonon kohta)
Task (FI: tehtävä), work item is splitted to tasks (of size at most 1..1,5 work days)
Planning poker (helps in estimation = ”smart quesses”)
Velocity, ability to complete tasks (FI: vauhti, kehitysnopeus, kehityskyvykkyys)
Scrum Master (FI: scrummaster, scrummasteri)
Product Owner (FI: tuoteomistaja)
Refactoring, fine-tuning (FI: refaktorointi, kodin jalostaminen, hiominen, parantelu)
Burndown Chart (or Burnup Chart) (FI: edistymiskäyrä, etenemiskaavio, edistymisen seuranta)
Definition of Done (DoD), how you know when it is ready ? (FI: valmiin määritelmä).



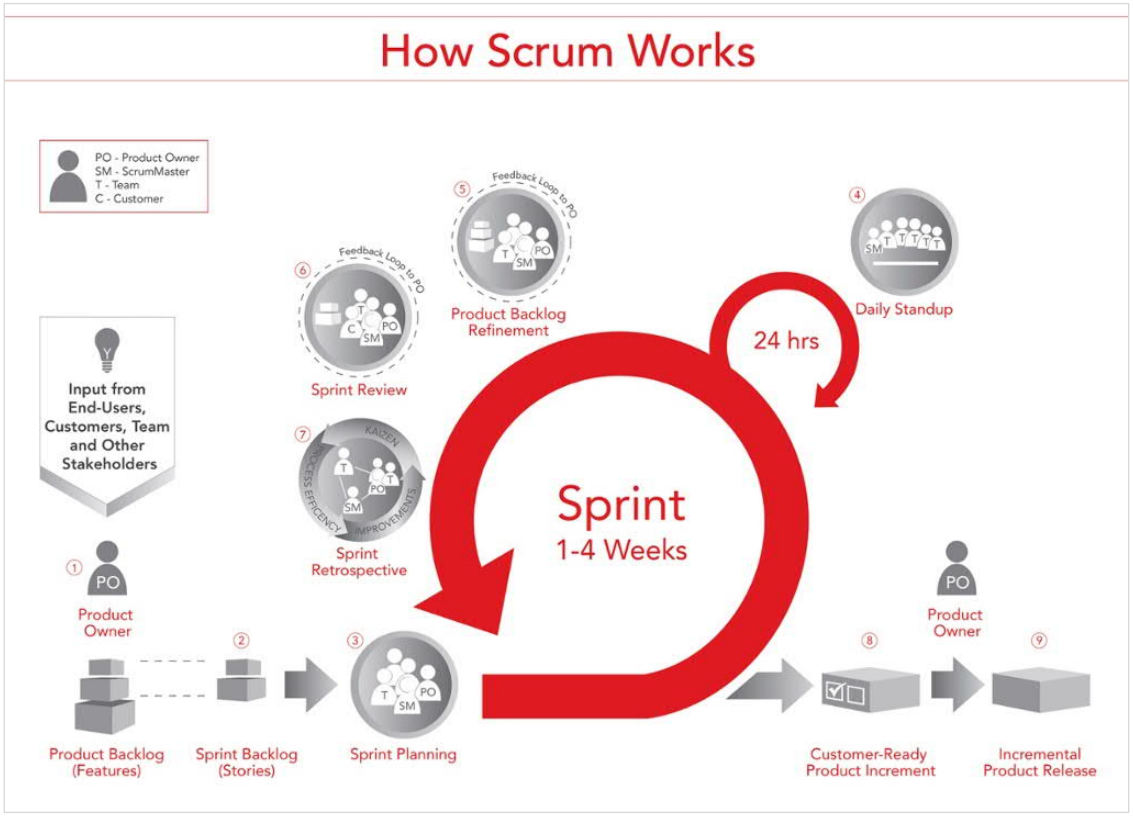
MoSCoW Prioritization

The MoSCoW Prioritization scheme derives its name from the first letters of the phrases

- “Must have”
- “Should have”
- “Could have”
- “Won’t have”.

The labels are in decreasing order of priority with “Must have” features being those without which the product will have no value and “Won’t have” features being those that, although they would be nice to have, are not necessary to be included.

[SBOK Guide 2016]

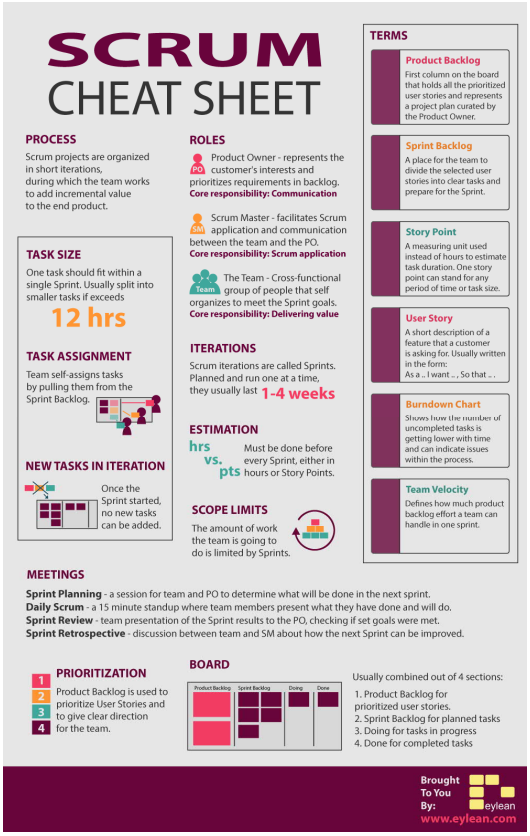


[www.thinglink.com]

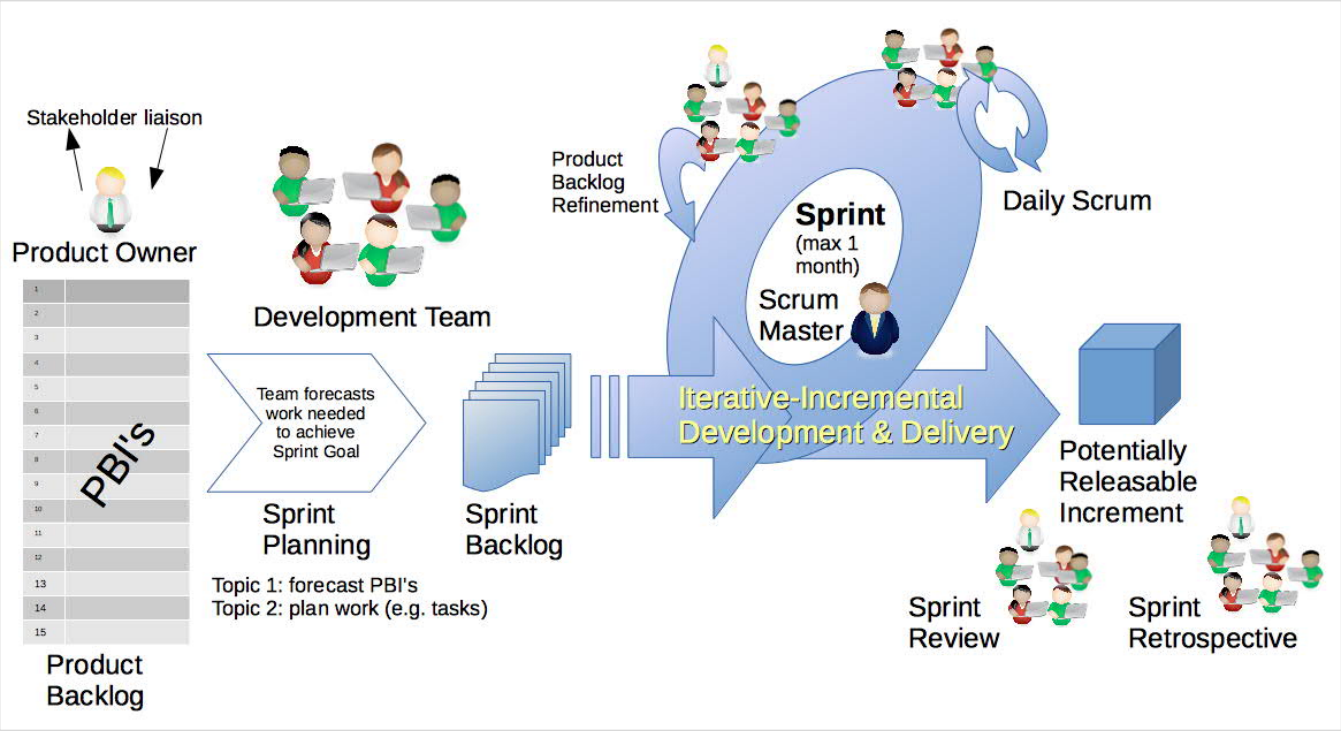
There are many agile and Scrum videos at YouTube, for example

Introduction to Scrum - 7 Minutes
https://www.youtube.com/watch?v=9TycLR0TqFA

Intro to Scrum in Under 10 Minutes
https://www.youtube.com/watch?v=XU0llRItyFM



Scrum



[www.sciodev.com]

Sprints (iterations) [Scrum Guide 2017]

Sprint Planning

The work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team.

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.

- What can be done this Sprint?
- How will the chosen work get done?
- Sprint Goal.

Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

Tampereen yliopisto
Tampere University

Sprints (iterations) [Scrum Guide 2017]

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team. The Daily Scrum is held every day of the Sprint. At it, the Development Team plans work for the next 24 hours. This optimizes team collaboration and performance by inspecting the work since the last Daily Scrum and forecasting upcoming Sprint work. The Daily Scrum is held at the same time and place each day to reduce complexity.

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

Notice: this daily meeting is for full-time teams (8 h workdays). Part-time teams may have “status meeting” 1..2 times in a week.

Tampereen yliopisto
Tampere University

Sprints (iterations) [Scrum Guide 2017]

The Sprint Review includes the following elements:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
- The Product Owner explains what Product Backlog items have been “Done” and what has not been “Done”;
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;
- The Development Team demonstrates the work that it has “Done” and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely target and delivery dates based on progress to date (if needed);
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning;
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality or capability of the product.

Tampereen yliopisto
Tampere University

Sprints (iterations) [Scrum Guide 2017]

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning.

This is at most a three-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose.

The purpose of the Sprint Retrospective is to:

• Inspect how the last Sprint went with regards to people, relationships, process, and tools;

• Identify and order the major items that went well and potential improvements;

• Create a plan for implementing improvements to the way the Scrum Team does its work.

Tampereen yliopisto
Tampere University

Scrum recommendations

- At the end of every working day, there should be a demoable product version available. If you don't get your work item finished (tested to work) during a working day, you do not merge it to version control, you continue with it on next working day.
- No overtime working on evenings or weekends.
- Ideally, team should have only one project at a time to work with.
- Team is supposed to be experienced, everybody knows each other (strenghts and weaknesses).

Tampereen yliopisto
Tampere University

Backlogs

Tampereen yliopisto
Tampere University

Backlogs (FI terminology may vary)

Product backlog, PB (FI: tuotepino)

- customer's and end-users' wishes, features, and functionalities, usually in form of user stories, in priority order

Sprint backlog, SB (FI: tehtäväpino)

- development team takes a few PB items to one iteration/sprint, and splits those to smaller parts (work items).

Warning; "work item" has not a solid meaning or explanation.

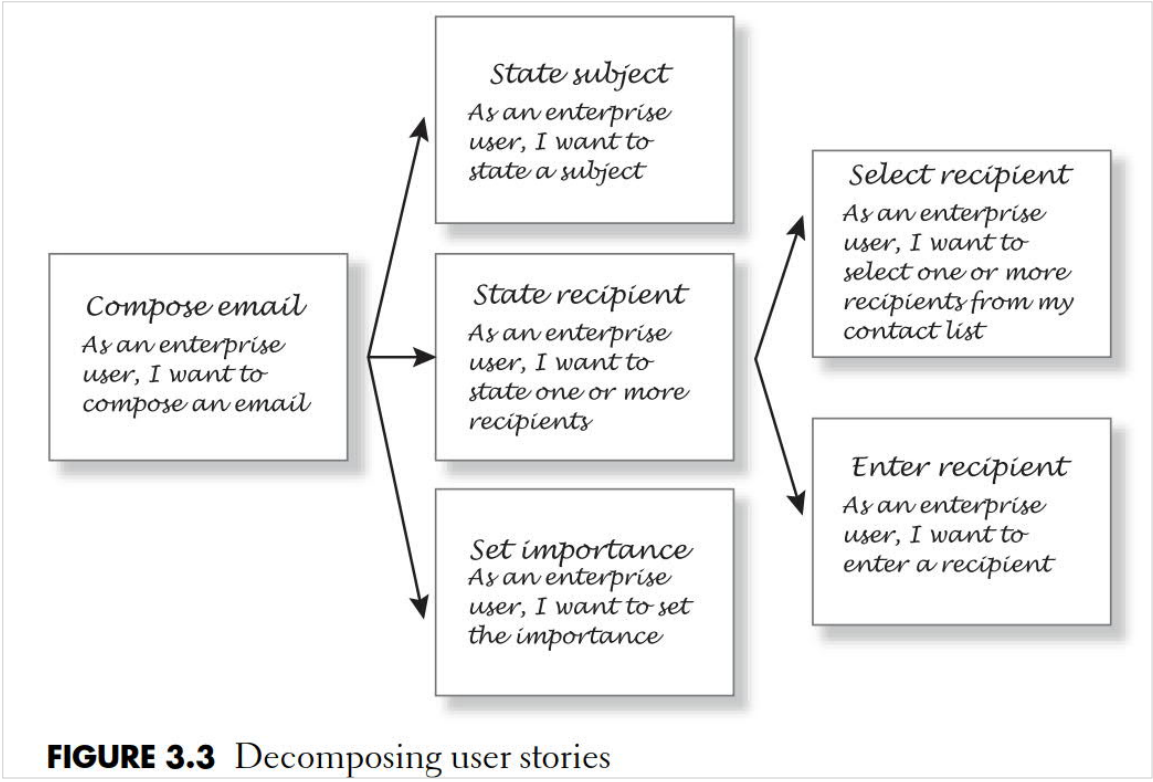


FIGURE 3.3 Decomposing user stories

[Agile product management, 2010]

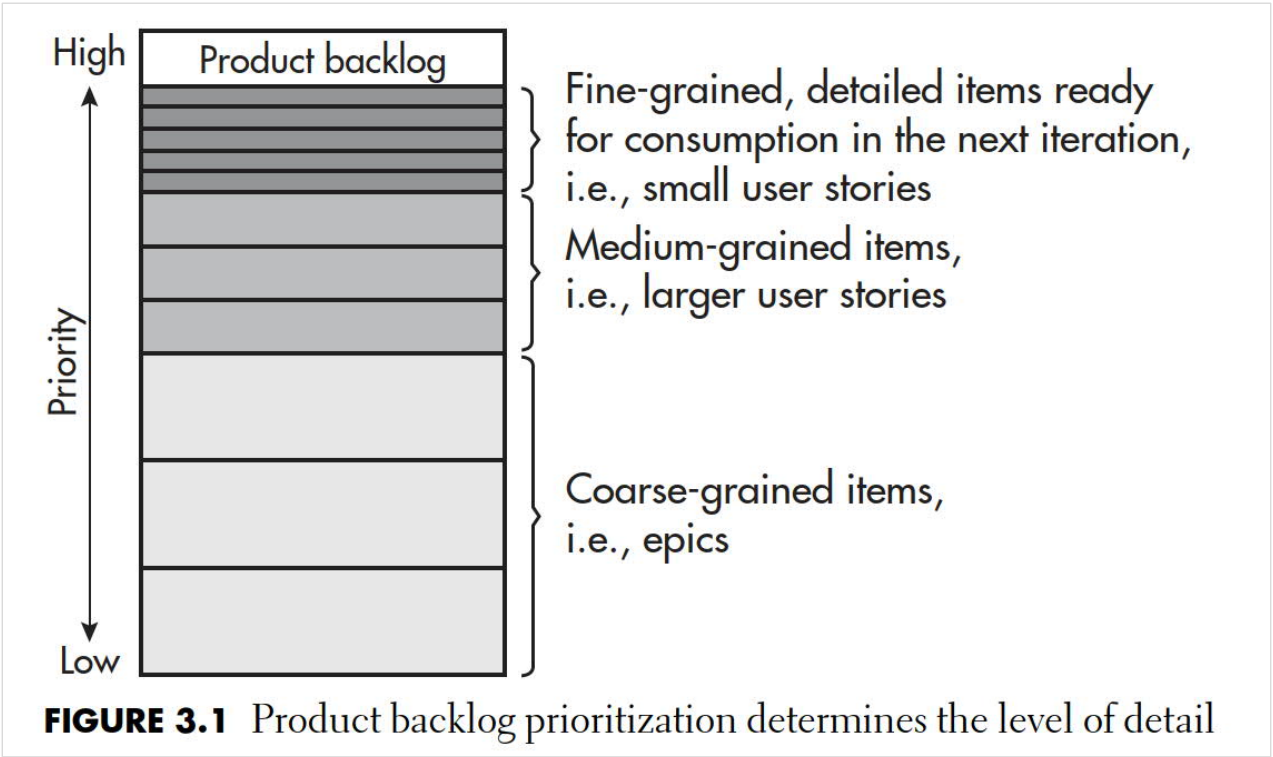


FIGURE 3.1 Product backlog prioritization determines the level of detail

[Agile product management, 2010]

Tampereen yliopisto
Tampere University

Product backlog [Scrum Guide 2017]

The Product Backlog is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

A Product Backlog is never complete. The earliest development of it lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. If a product exists, its Product Backlog also exists.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases.

Multiple Scrum Teams often work together on the same product. One Product Backlog is used to describe the upcoming work on the product.

Higher ordered (higher priority) Product Backlog items are usually clearer and more detailed than lower ordered ones.

Tampereen yliopisto
Tampere University

Sprint backlog [Scrum Guide 2017]

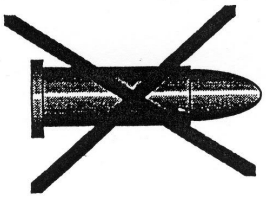
The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a “Done” Increment.

The Sprint Backlog makes visible all the work that the Development Team identifies as necessary to meet the Sprint Goal. To ensure continuous improvement, it includes at least one high priority process improvement identified in the previous Retrospective meeting.

The Sprint Backlog is a plan with enough detail that changes in progress can be understood in the Daily Scrum. The Development Team modifies the Sprint Backlog throughout the Sprint, and the Sprint Backlog emerges during the Sprint. This emergence occurs as the Development Team works through the plan and learns more about the work needed to achieve the Sprint Goal.

One SB work item should be of size that could be done in 1..1,5 days.

There is no silver bullet!



No tool, no method will save you from having to think!

Tampereen yliopisto
Tampere University

Scrumboard

Tampereen yliopisto
Tampere University

Scrumboard

Scrumboard is a tool used by the Scrum Team to plan and track progress during each Sprint.

The Scrumboard contains four columns to indicate the progress of the estimated tasks for the Sprint:

- a To Do column for tasks not yet started,
- an In Progress column for the tasks started but not yet completed,
- a Testing column for tasks completed but in the process of being tested, and
- a Done column for the tasks that have been completed and successfully tested.

[SBOK Guide 2016]

Instead of that, a kanban board (TODO, DOING, DONE) is often used. Then you have to figure how to present PB and SB (lists ?).

Tampereen yliopisto
Tampere University

Scrum board (kanban style)

Typically, the board is divided into horizontal lanes or vertical columns that the team use to track the progress of the agreed-upon work to be completed in the sprint.

A scrum board may include these columns:

- To Do: the prioritized backlog of product features planned for the current sprint
- In Progress: the list of tasks that have been started
- In Test: completed tasks that are being tested for verification
- Done: tasks that have been completed and verified by testing.

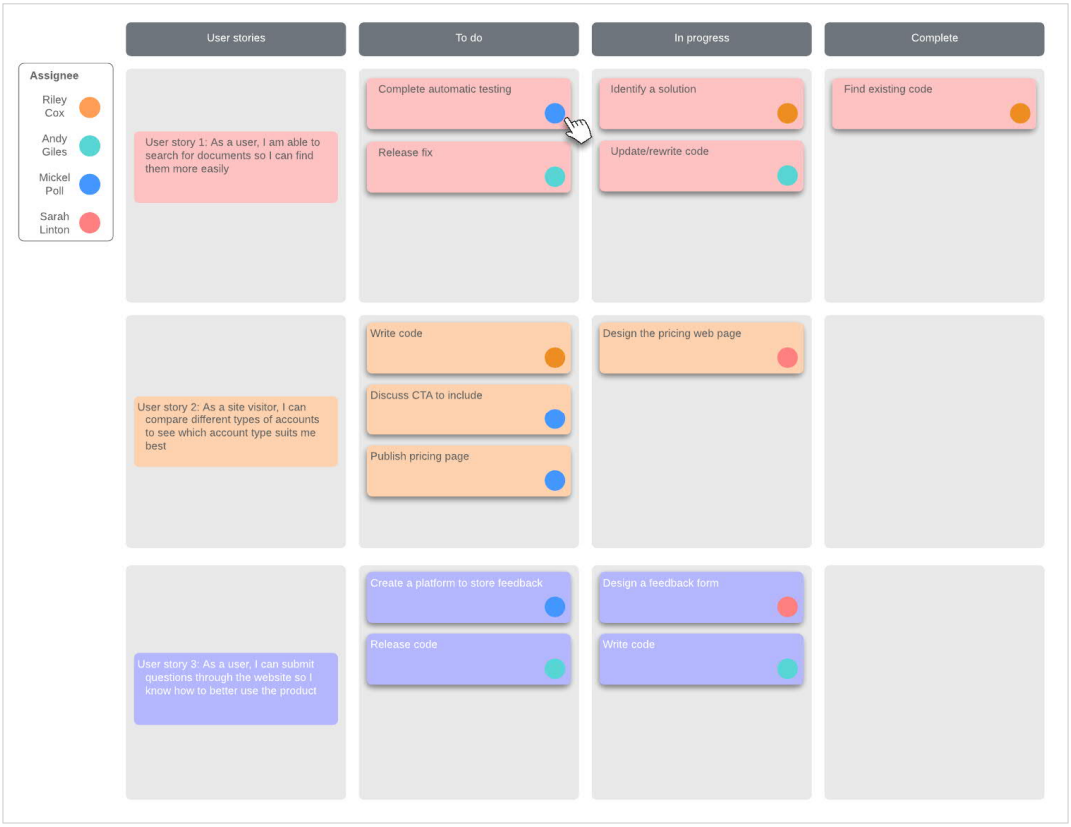
You can add or remove lanes and columns to suit your needs.

[https://www.lucidchart.com/blog/kanban-vs-scrum]

Tampereen yliopisto
Tampere University

Scrum task board example

[https://www.lucidchart.com/blog/kanban-vs-scrum]



Tampereen yliopisto
Tampere University

Why use a scrum board

A couple of important principles of scrum are teamwork and transparency. All members of the team need to be aware of the work being done, the team members completing the work, the progress of that work, and team accomplishments.

A scrum board facilitates these principles in the following ways:

Promotes team interaction and discussion. Team members and stakeholders look at the board throughout the day to discuss progress and to prioritize work.

Makes information visual and easily accessible. Whether your board is physical or virtual, the information it displays is easy to digest and easy to access. Anybody looking at the board can quickly assess where the team is in the iteration and what still needs to be done to achieve goals.

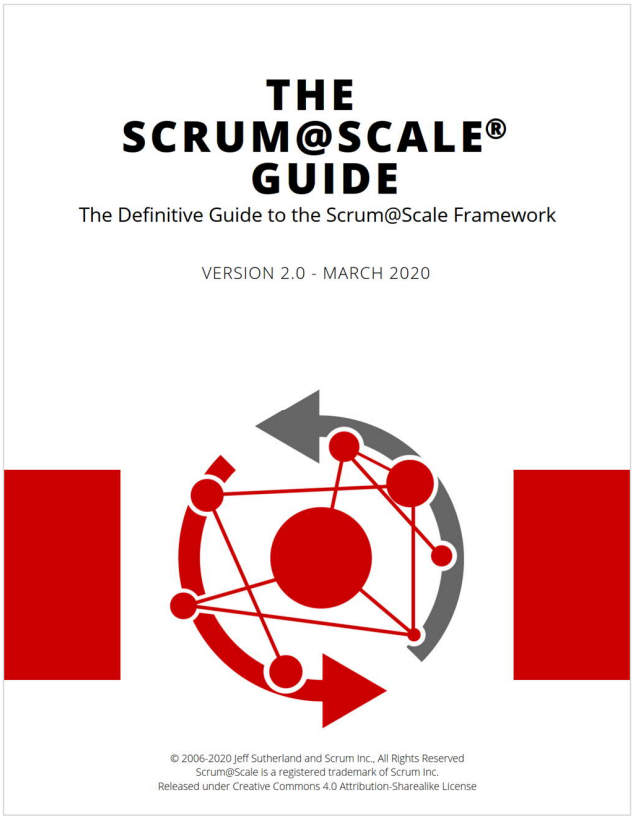
Supports full team commitment. When the team sees all the tasks on the board, it keeps them from focusing only on individual tasks. As individual tasks are completed, those with the time step up and take on additional tasks.

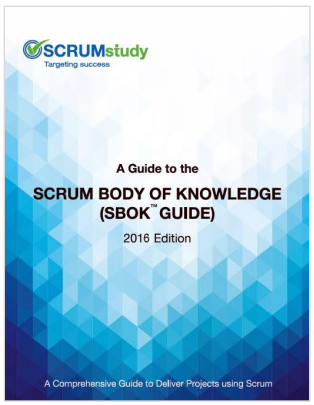
Scrum boards are designed specifically for completing specific user stories and tasks within a specific time frame. You'll want to use a scrum board if your team develops based on iterations or sprints. [https://www.lucidchart.com/blog/kanban-vs-scrum]

[https://www.lucidchart.com/blog/kanban-vs-scrum]

Scrum of scrums

Scrum of Scrums





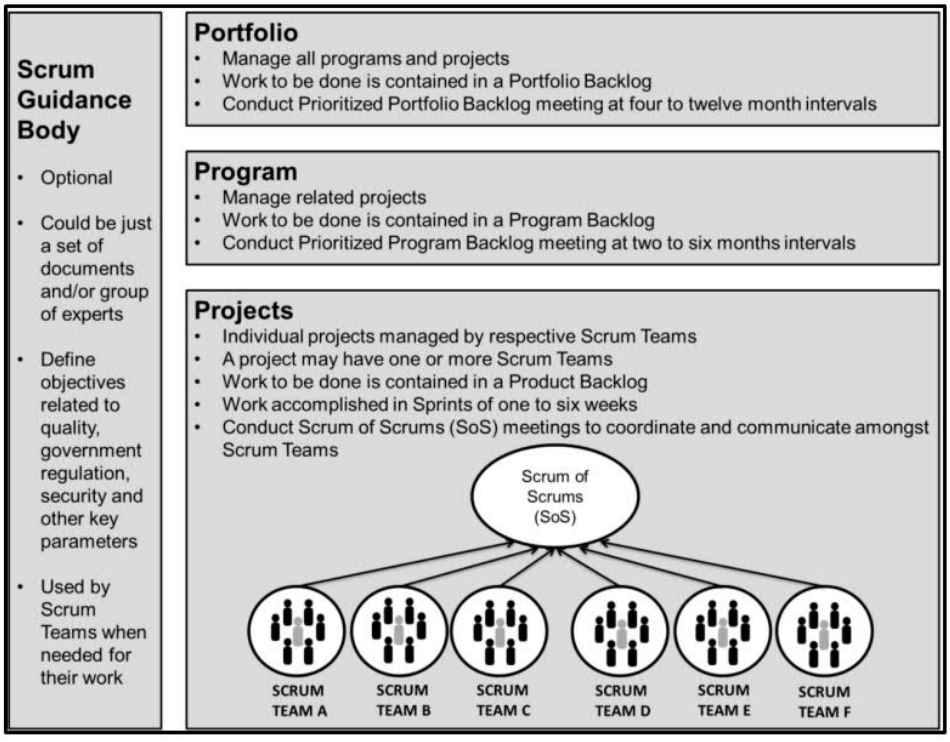
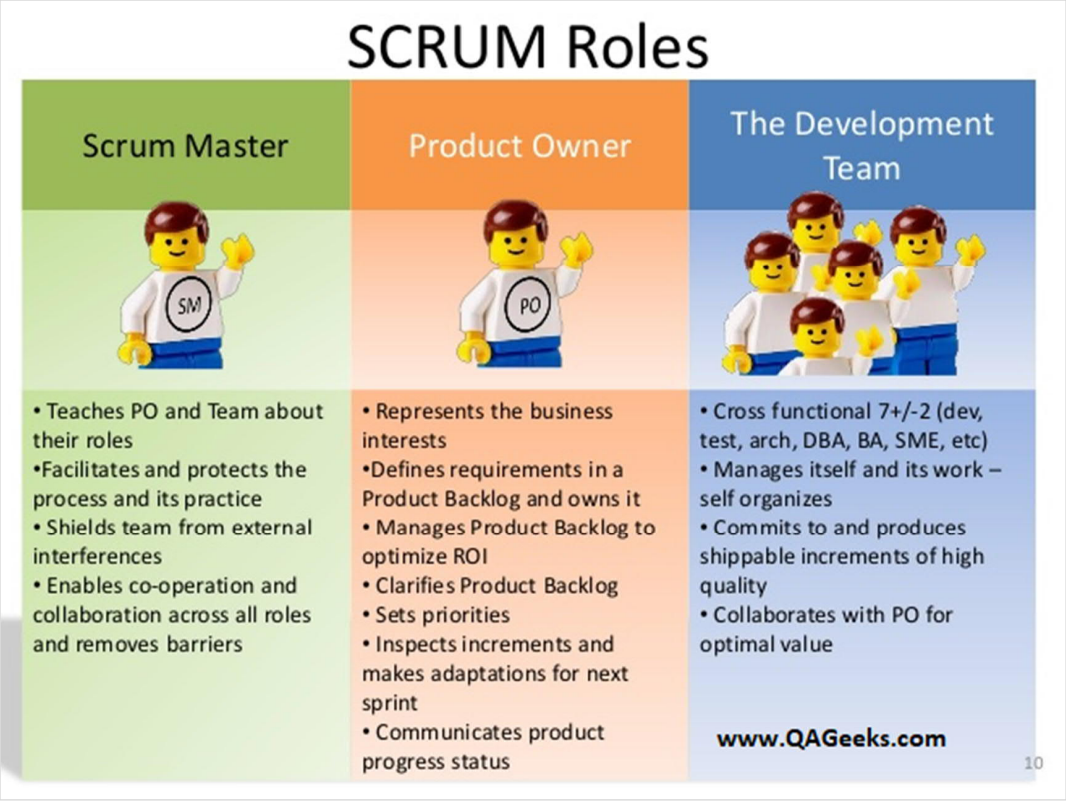


Figure 3-5: Scrum Across the Organization for Projects, Programs, and Portfolios

Scrum roles


SCRUM Roles
Scrum Master
Product Owner
The Development Team
SM
PO
• Teaches PO and Team about their roles
•Facilitates and protects the process and its practice
• Shields team from external interferences
• Enables co-operation and collaboration across all roles and removes barriers
• Represents the business interests
•Defines requirements in a Product Backlog and owns it
• Manages Product Backlog to optimize ROI
• Clarifies Product Backlog
• Sets priorities
• Inspects increments and makes adaptations for next sprint
• Communicates product progress status
• Cross functional 7+/-2 (dev, test, arch, DBA, BA, SME, etc)
• Manages itself and its work – self organizes
• Commits to and produces shippable increments of high quality
• Collaborates with PO for optimal value
www.QAGeeks.com
10

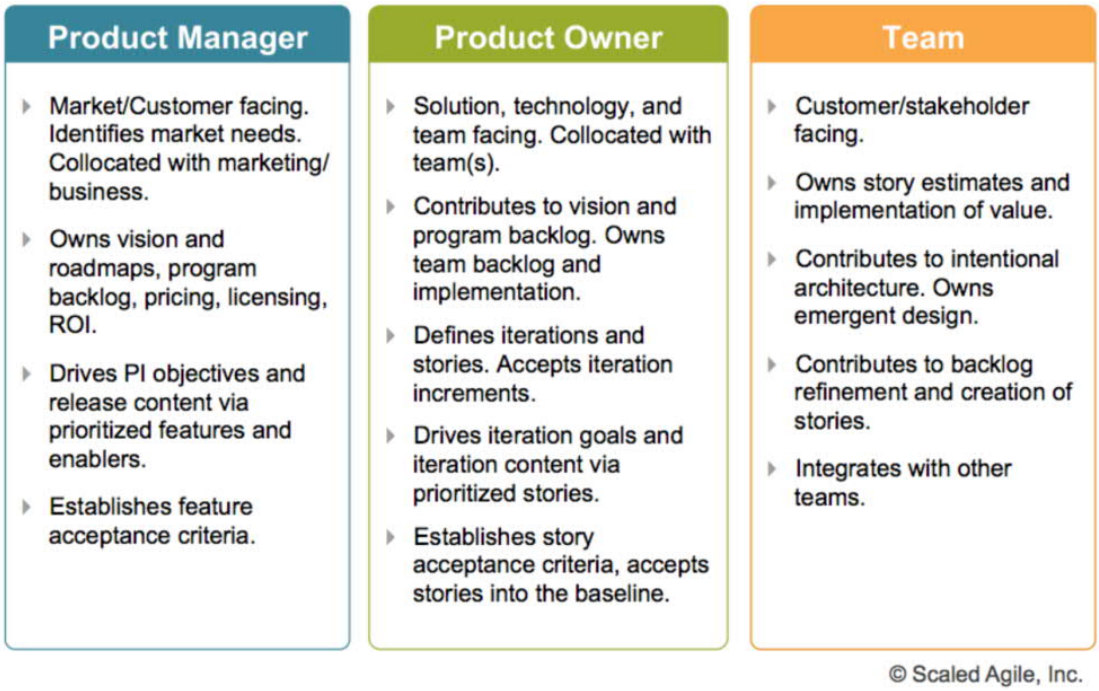


Figure 1. Release content governance

...or scrum master will be the scrum project scapegoat.

Tampereen yliopisto
Tampere University

Scrum team [Scrum Guide 2017]

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of “Done” product ensure a potentially useful version of working product is always available.

- - -

In Finland, often Scrum Master means, or does the same work than, a Project Manager. Which is actually against Scrum Guide.

Scrum team is supposed to be an experienced TEAM.

Tampereen yliopisto
Tampere University

Scrum Team [Scrum Guide 2017]

Development Teams have the following characteristics:

• They are self-organizing. No one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality;

• Development Teams are cross-functional, with all the skills as a team necessary to create a product Increment;

• Scrum recognizes no titles for Development Team members, regardless of the work being performed by the person;

• Scrum recognizes no sub-teams in the Development Team, regardless of domains that need to be addressed like testing, architecture, operations, or business analysis; and,

• Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team as a whole.

Tampereen yliopisto
Tampere University

Scrum development Team size

[Scrum Guide 2017]

Optimal Development Team size is small enough to remain nimble and large enough to complete significant work within a Sprint. Fewer than three Development Team members decrease interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment.

Having more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to be useful.

The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

Tampereen yliopisto
Tampere University

Product owner (PO)

As described in the Scrum Guide, a Scrum Product Owner is responsible for maximizing the value of the product resulting from the work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

The Product Owner is the sole person responsible for managing the Product Backlog. Product Backlog management includes:

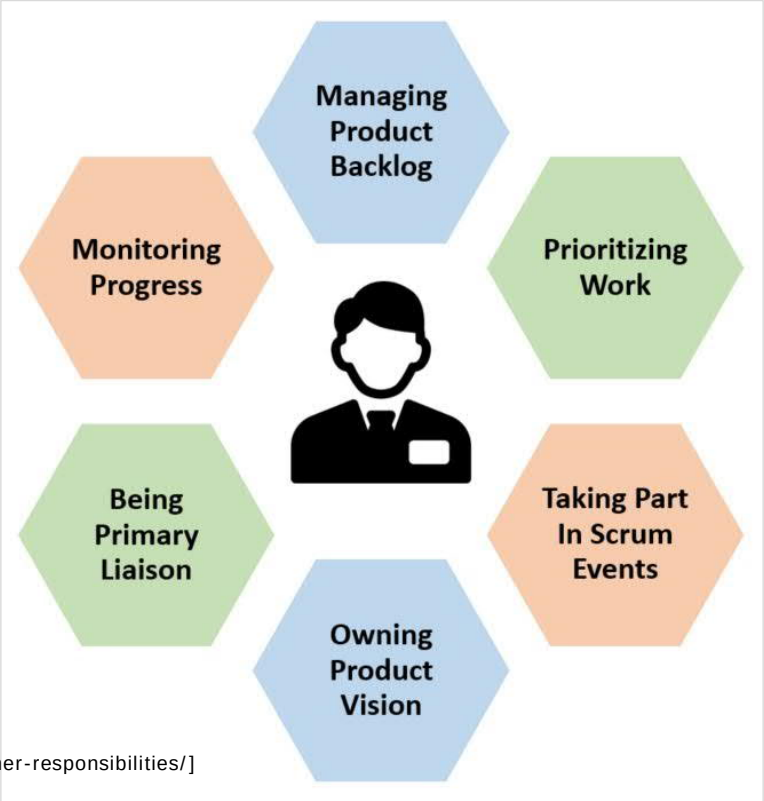
- Clearly expressing Product Backlog items.
- Ordering the items in the Product Backlog to best achieve goals and missions.
- Optimizing the value of the work the Development Team performs.
- Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next.
- Ensuring the Development Team understands items in the Product Backlog to the level needed.

The Product Owner may do the above work or have the Development Team do it. However, the Product Owner remains accountable.

[Scrum Guide 2017]

Tampereen yliopisto
Tampere University

Product owner responsibilities



[https://productmint.com/a-quick-list-of-product-owner-responsibilities/]

Tampereen yliopisto
Tampere University

Scrum Master, 1 / 3 [Scrum Guide 2017]

Scrum Master Service to the Product Owner

The Scrum Master serves the Product Owner in several ways, including:

- Ensuring that goals, scope, and product domain are understood by everyone on the Scrum Team as well as possible;
- Finding techniques for effective Product Backlog management;
- Helping the Scrum Team understand the need for clear and concise Product Backlog items;
- Understanding product planning in an empirical environment;
- Ensuring the Product Owner knows how to arrange the Product Backlog to maximize value;
- Understanding and practicing agility;
- Facilitating Scrum events as requested or needed.

Tampereen yliopisto
Tampere University

Scrum Master, 2 / 3 [Scrum Guide 2017]

Scrum Master Service to the Development Team

The Scrum Master serves the Development Team in several ways, including:

- Coaching the Development Team in self-organization and cross-functionality;
- Helping the Development Team to create high-value products;
- Removing impediments to the Development Team's progress;
- Facilitating Scrum events as requested or needed; and,
- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.

Tampereen yliopisto
Tampere University

Scrum Master, 3 / 3 [Scrum Guide 2017]

Scrum Master Service to the Organization

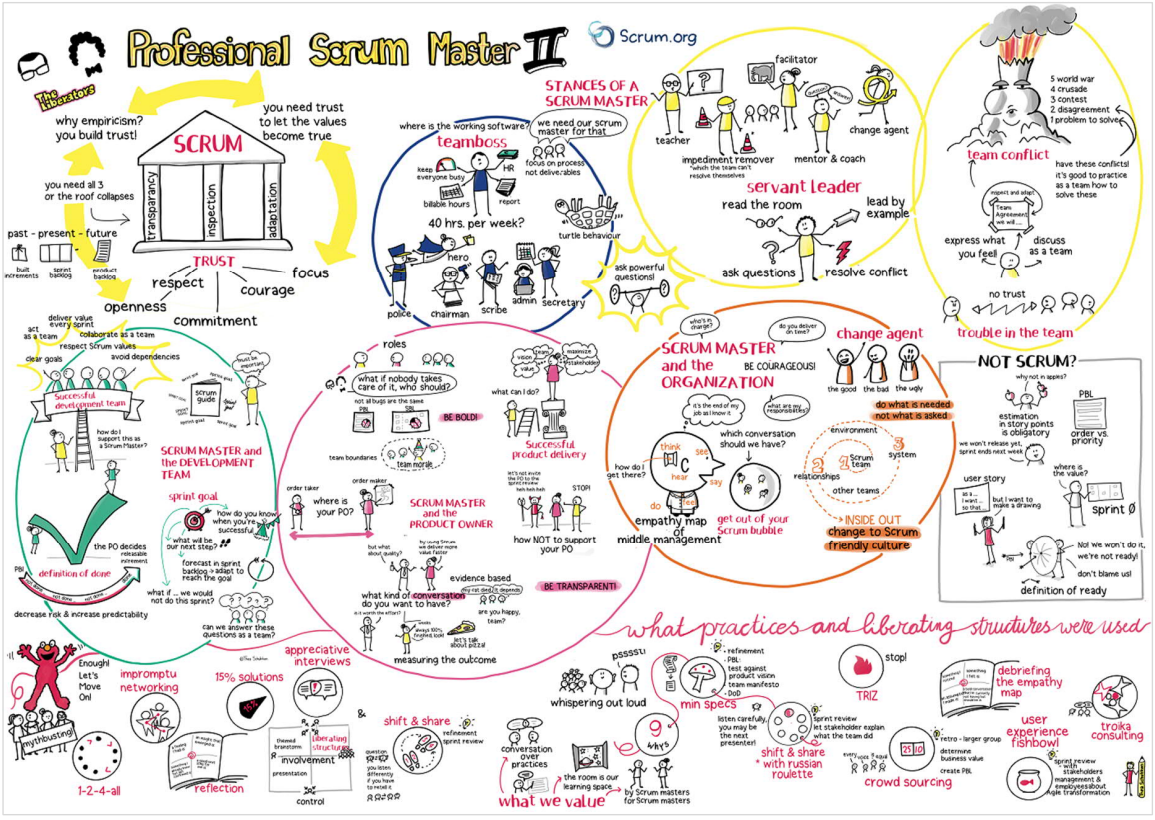
The Scrum Master serves the organization in several ways, including:

- Leading and coaching the organization in its Scrum adoption;
- Planning Scrum implementations within the organization;
- Helping employees and stakeholders understand and enact Scrum and empirical product development;
- Causing change that increases the productivity of the Scrum Team; and,
- Working with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization.

Tampereen yliopisto
Tampere University

A quick overview of Scrum Master tasks and skill sets

[https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/PSMII_2.png]



Tampereen yliopisto
Tampere University

Scrum Master

The scrum master is the team role responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use.

The responsibilities of this role include:

- clearing obstacles
- establishing an environment where the team can be effective
- addressing team dynamics
- ensuring a good relationship between the team and product owner as well as others outside the team
- protecting the team from outside interruptions and distractions.

Burndown / burnup charts

Tampereen yliopisto
Tampere University

Burndown / burnup chart

The team displays, somewhere on a wall of the project room, a large graph relating the quantity of work remaining (on the vertical axis) and the time elapsed since the start of the project (on the horizontal, showing future as well as past). This constitutes an "information radiator", provided it is updated regularly. Two variants exist, depending on whether the amount graphed is for the work remaining in the iteration ("sprint burndown") or more commonly the entire project ("product burndown").

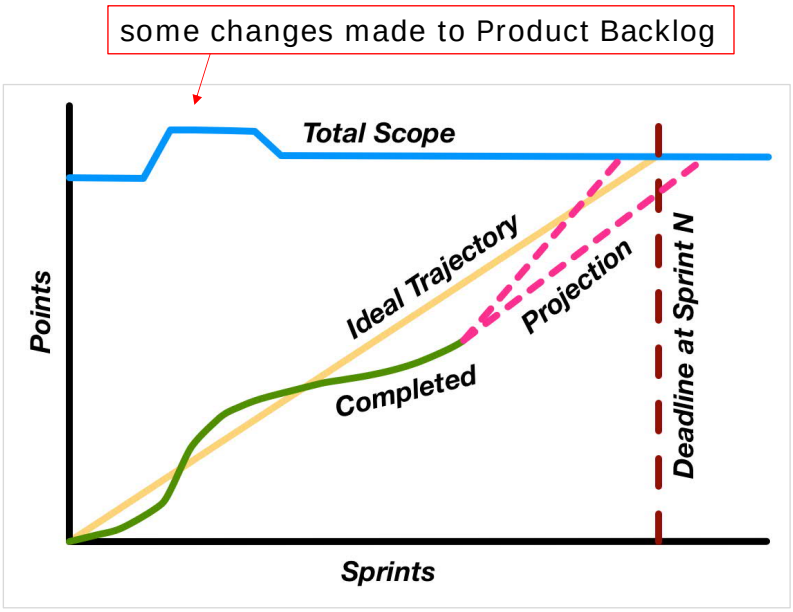
The term "burn chart" is sometimes encountered, possibly as a generalization covering variants such as the "burn up chart".

This practice results in up-to-date project status being not only visible, but in fact shoved into the faces of everyone involved: as a result it encourages the team to confront any difficulties sooner and more decisively.

The original chart was Burn-Up, but you may also use Burn-Down chart, to show how little work is left, still to be done.

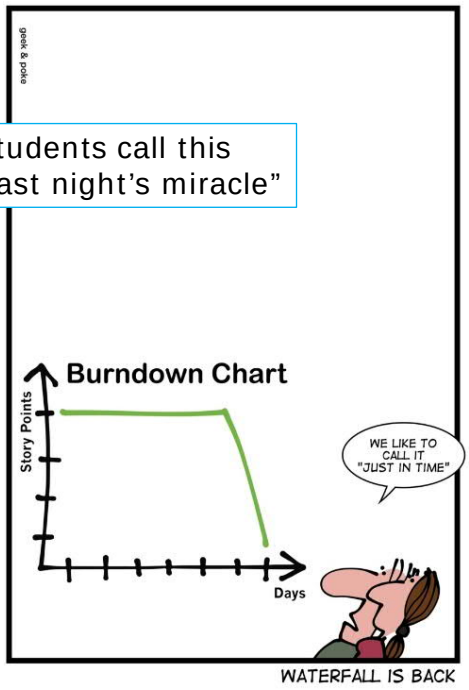
Burnup chart example, 1

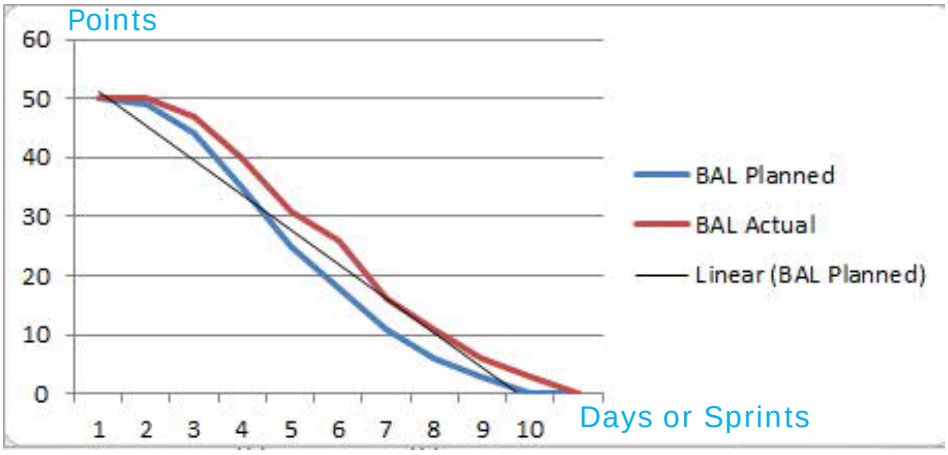
- Horizontal (X) Axis – Amount of Time (in sprints)
- Vertical (Y) Axis – Amount of Work to be Done (in points)
- Blue Line – Total points; the overall scope of the project over time.
- Green Line – Completed points; the team's progress towards their goal over time.
- Brown Dotted Line – A budget or calendar deadline.
- Yellow Line – The ideal pace needed to meet the deadline.
- Pink Dotted Line – A high and low projection of likely future progress, based on historical velocity data. (Optional)



Burndown chart, 1

Students call this "last night's miracle"



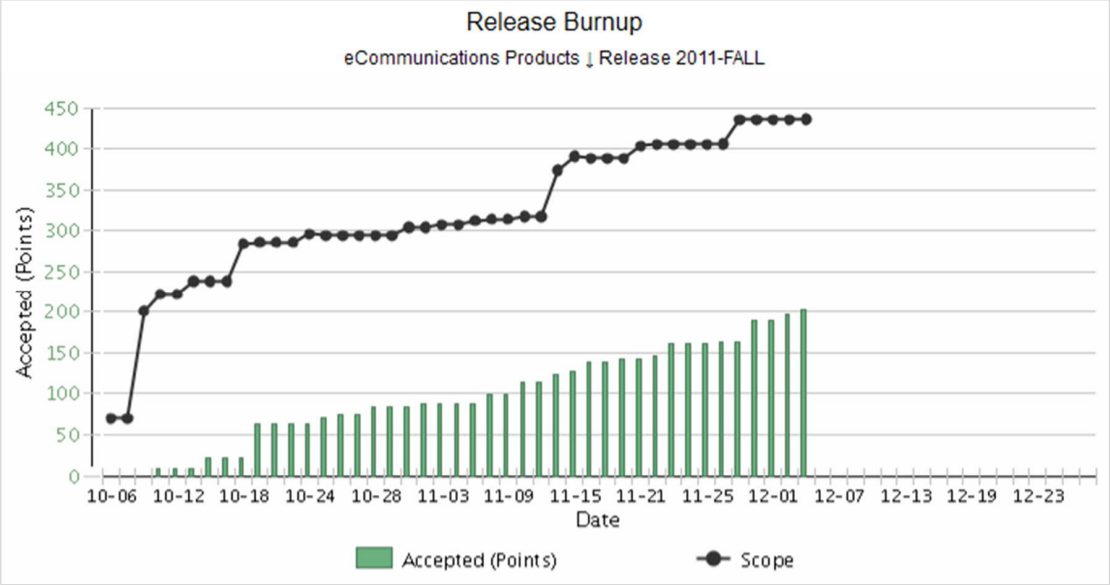


[https://qualitasrebus.com/]

Tampereen yliopisto
Tampere University

Just because you are Agile, does not mean you eliminated scope creep.

We focus on the features that are highest in priority and deliver early and often to create a minimal marketable product (MMP) giving us a serious advantage over other methodologies that struggle to understand the WHOLE PROJECT from the very beginning.



[myagilemind.wordpress.com/2011/12/05/scope-creep-agile-is-not-immune-to-it/]

Tampereen yliopisto
Tampere University

Velocity

At the end of each iteration, the team adds up effort estimates associated with user stories that were completed during that iteration. This total is called velocity.

Knowing velocity, the team can compute (or revise) an estimate of how long the project will take to complete, based on the estimates associated with remaining user stories and assuming that velocity over the remaining iterations will remain approximately the same. This is generally an accurate prediction, even though rarely a precise one.

“Worked example:” an agile team has started work on an iteration, planning to complete stories A and B, estimated at 2 points each, and story C, estimated at 3 points. At the end of the iteration, stories A and B are 100% complete but C is only 80% complete.

Agile teams generally acknowledge only two levels of completion, 0% done or 100% done. Therefore, C is not counted toward velocity, and velocity as of that iteration is 4 points.

Suppose the user stories remaining represent a total of 40 points; the team’s forecast of the remaining effort for the project is then 10 iterations.

Usually the team’s velocity will grow along the project.

Wrong use of agile or Scrum

Wrong use of agile or Scrum

”Scrum is not a silver bullet”

Agile methods do not fit all software development projects.

Most common problems in Scrum software projects

- too many and too big chunks of work (items) in SB
- inexperienced developers (”hey try this new agile method”)
- somebody tolds team what to do (motivation)
-
-

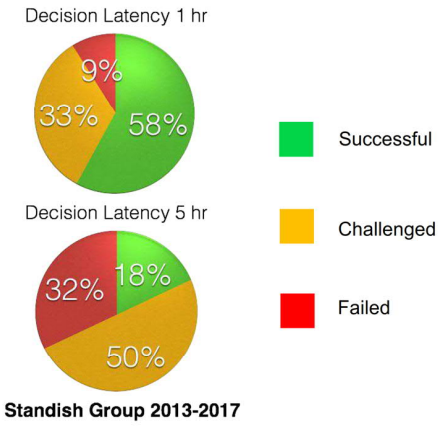
Scrum@Scale Enables Focus Across the Organization

Reduce WIP, eliminate dark work, and focus on value / outcomes

25% of staff delivering stories customers will use

64% x 70% = **45% of staff** delivering stories the customer will never or rarely use **(Standish Group)**

Typically 30% of staff working on zero value stories



Standish Group 2013-2017

rce: Dr J Sutherland, Scrum.inc based on research by the Standish Group

Lack of direction causes staff to make up work.
Unwillingness to prioritize proliferates useless projects.
Delayed decision-making is the primary driver of project failure and budget overrun.

scruminc.

Scrum, wrong use

(anti-patterns = wrong habits)

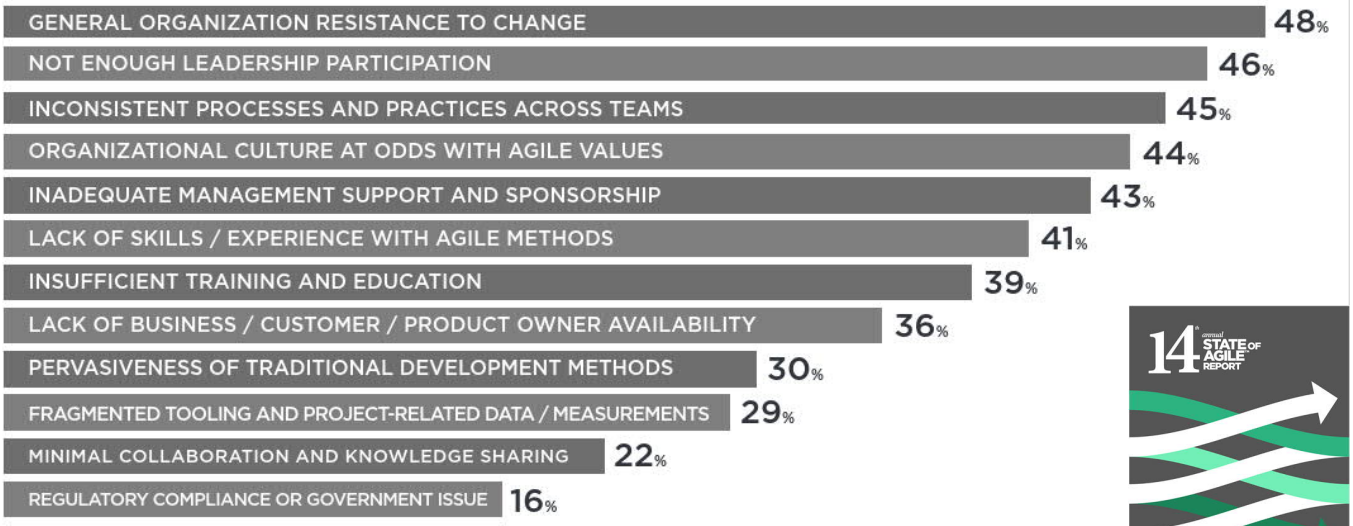
[Veli-Pekka Eloranta, Kai Koskimies,Tommi Mikkonen and Jyri Vuorinen:
Scrum Anti-patterns – An Empirical Study, 2013]

Table IX. Summary of Scrum anti-patterns in different companies

Name	Company size			Scrum Experience		
	S	M	L	<1	<2	2+
Too long sprint		2	2	1		3
Testing in next sprint	1	1	4	1	1	4
Big requirements document		1	3	1		3
Customer product owner	1	1	2		2	2
Product owner without authority		2	3	1		4
Unordered product backlog	1	1	3	1	1	3
Work estimates given to teams		1	1	1		1
Invisible progress	1		3		1	3
Customer caused disruption	2	2	4	1	3	4
Semi-functional teams		1	2	1		2

CHALLENGES EXPERIENCED WHEN ADOPTING & SCALING AGILE

The top three responses cited as challenges/barriers to adopting and scaling Agile practices indicate that internal culture remains an obstacle for success in many organizations.



*Respondents were able to make multiple selections

Tampereen yliopisto
Tampere University

Scrum-BUT

Tampereen yliopisto
Tampere University

The most popular: Scrum-BUT

The most used Scrum method in Finland (and in the world) is Scrum-BUT.

If you ask some company ”Are you using Scrum ?”, they will usually reply ”Yes, we are using Scrum... but we have made a few shortcuts and modifications.”

So, Scrum-BUT means modified Scrum. Adapt ”best practices” from all public sources you see or hear, and make your own method combination which suits best your work.

When James Coplien had Scrum Master and Product Owner traing courses at TUT, he said that ”If you are using agile/Scrum by the book, you are not agile”.

Tampereen yliopisto
Tampere University

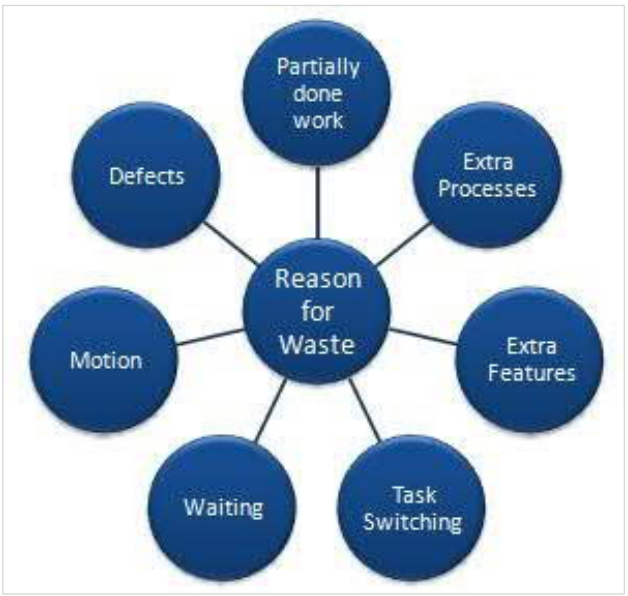
Lean

Lean seven principles

”concentrate to the essentials”

[https://hangoutagile.com/wp-content/uploads/2018/09/Lean-Software-Development.jpg]

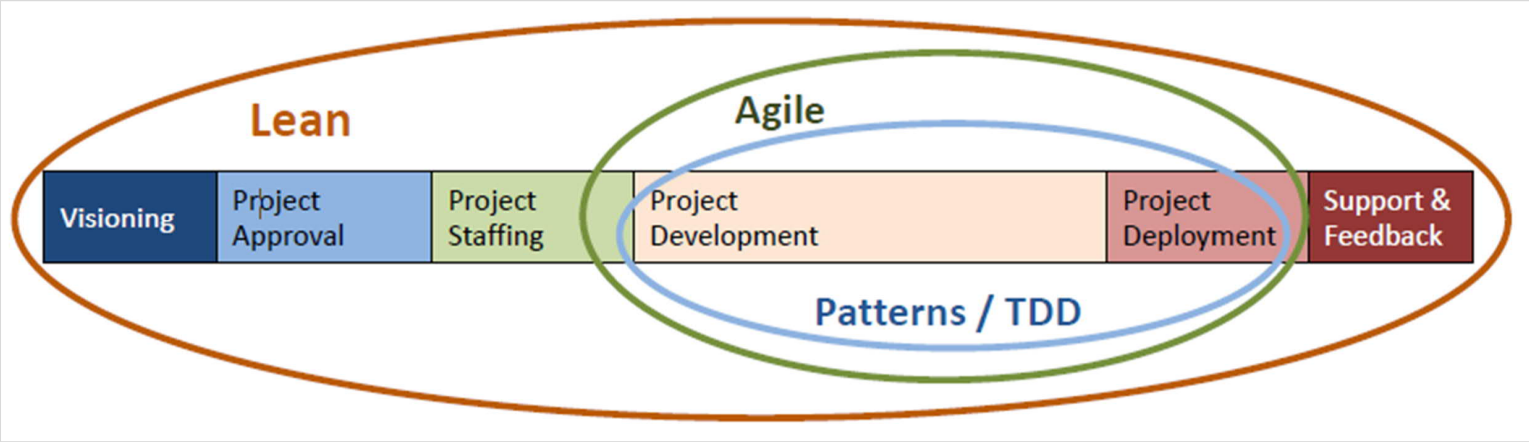
Lean



Reason for waste

Eliminating waste

[Agile Lean Development for Waste Management: A Review, 2017]



TDD = Test Driven Development

[https://www.infoq.com/news/2008/11/Lean-Agile-Alan-Shalloway/]

Minimum viable product (MVP)

Tampereen yliopisto
Tampere University

MVP = minimum viable product

A common misunderstanding is that MVP is ”the lousiest product which developer dares to send to customer”.

The MVP is that version of the product that enables a full turn of the Build-Measure-Learn loop with a minimum amount of effort and the least amount of development time. The minimum viable product lacks many features that may prove essential later on.

However, in some ways, creating a MVP requires extra work: we must be able to measure its impact. For example, it is inadequate to build a prototype that is evaluated solely for internal quality by engineers and designers. We also need to get it in front of potential customers to gauge their reactions. We may even need to try selling them the prototype, as we'll soon see.

A minimum viable product (MVP) helps entrepreneurs start the process of learning as quickly as possible. It is not necessarily the smallest product imaginable, though; it is simply the fastest way to get through the Build-Measure-Learn feedback loop with the minimum amount of effort.

Contrary to traditional product development, which usually involves a long, thoughtful incubation period and strives for product perfection, the goal of the MVP is to begin the process of learning, not end it. Unlike a prototype or concept test, an MVP is designed not just to answer product design or technical questions. Its goal is to test fundamental business hypotheses.

[Eric Ries: Lean startup, 2011]

Tampereen yliopisto
Tampere University

MVP = minimum viable product

A common misunderstanding is that MVP is ”the lousiest product which developer dares to send to customer”.

A minimum viable product (MVP) is a concept from Lean Startup that stresses the impact of learning in new product development. Eric Ries, defined an MVP as that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort. This validated learning comes in the form of whether your customers will actually purchase your product.

A key premise behind the idea of MVP is that you produce an actual product (which may be no more than a landing page, or a service with an appearance of automation, but which is fully manual behind the scenes) that you can offer to customers and observe their actual behavior with the product or service. Seeing what people actually do with respect to a product is much more reliable than asking people what they would do.

The primary benefit of an MVP is you can gain understanding about your customers' interest in your product without fully developing the product. The sooner you can find out whether your product will appeal to customers, the less effort and expense you spend on a product that will not succeed in the market.

[www.agilealliance.org/agile101/agile-glossary/]

Lean Startup; MVP

- A minimum viable product (MVP) helps entrepreneurs start the process of learning as quickly as possible. It is not necessarily the smallest product imaginable, though; it is simply the fastest way to get through the Build-Measure-Learn feedback loop with the minimum amount of effort.
- Contrary to traditional product development, which usually involves a long, thoughtful incubation period and strives for product perfection, the goal of the MVP is to begin the process of learning, not end it. Unlike a prototype or concept test, an MVP is designed not just to answer product design or technical questions.
- Its goal is to test fundamental business hypotheses.

- MVP is NOT the simplest sw version that customer accepts. "MVP" is often misused and misunderstood.

MVP definitions, 1

MVP, the minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort. [Eric Ries, 2009]

- so getting/gathering feedback is important.

A Minimum Viable Product Is Not a Product, It's a Process. An MVP is not just a product with half of the features chopped out, or a way to get the product out the door a little earlier. In fact, the MVP doesn't have to be a product at all. And it's not something you build only once, and then consider the job done. [Jim Brickman, 2016]

A minimum viable products are considered a risk reduction tool, that exist to help start-ups test the water without producing something that's not economically viable.

[BL, Business & IP Centre]

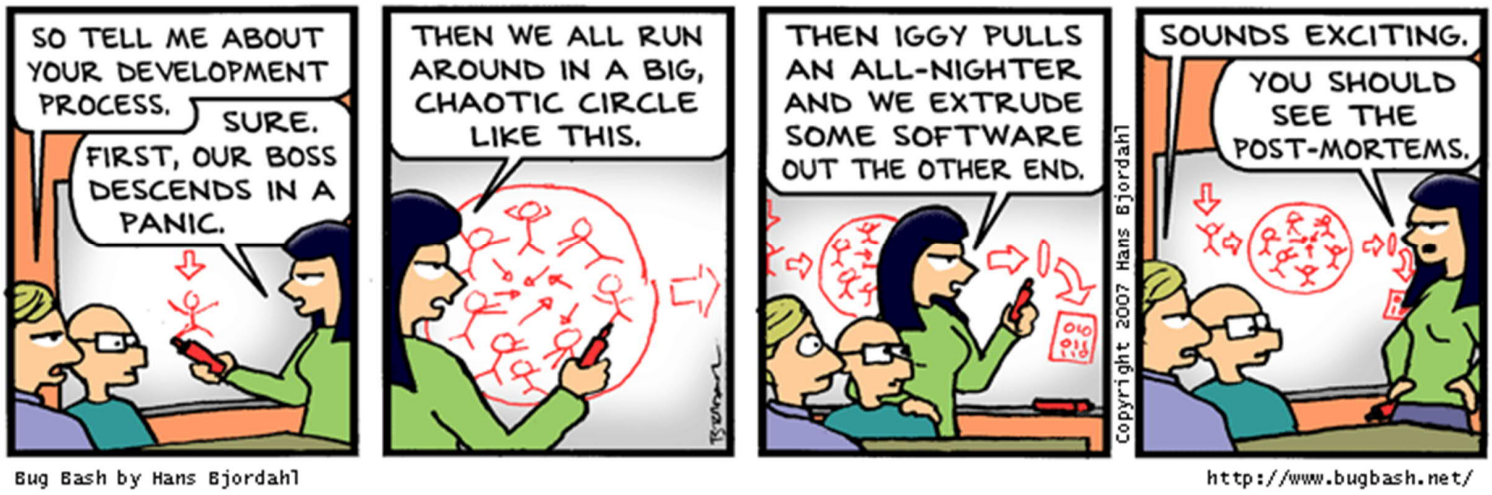
Lean Canvas

My First Product - My First Canvas

PROBLEM List your customer's top 3 problems **EXISTING ALTERNATIVES** List how these problems are solved today	**SOLUTION** Outline a possible solution for each problem **KEY METRICS** List the key numbers that tell you how your business is doing	**UNIQUE VALUE PROPOSITION** Single, clear, compelling message that turns an unaware visitor into an interested prospect **HIGH-LEVEL CONCEPT** List your X for Y analogy (e.g. YouTube = Flickr for videos)	**UNFAIR ADVANTAGE** Something that can not be easily copied or bought **CHANNELS** List your path to customers	**CUSTOMER SEGMENTS** List your target customers and users **EARLY ADOPTERS** List the characteristics of your ideal customers
COST STRUCTURE List your fixed and variable costs		**REVENUE STREAMS** List your sources of revenue		

Tampereen yliopisto
Tampere University

Some development process makes life easier, instead of ”desperate hacking”



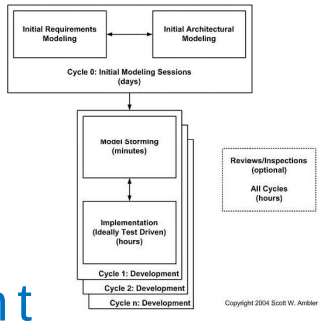
Agile model driven development (AMDD)

AMDD = agile model driven development

Agile Modeling

AMDD

Agile Model Driven Development (AMDD) is the agile version of Model Driven Development (MDD). The figure below overviews the fundamentals of AMDD:
- At the beginning of a project invest some time to do some initial modeling, particularly to explore the fundamental requirements and to identify a potential architectural approach.
- During development cycles you will typically model for several minutes and then implement for several hours or even days until you need to do some more modeling.
- The AM practices of Model With Others and Collective Ownership are effectively model reviews in process, removing the need for formal reviews.



Agile Modeling

Values

The Values of AM
- Communication
- Courage
- Feedback
- Humility
- Simplicity

Principles

Core Principles of AM
- Assume Simplicity
- Embrace Change
- Enabling the Next Effort is Your Secondary Goal
- Incremental Change
- Maximize Stakeholder Investment
- Model With a Purpose
- Multiple Models
- Quality Work
- Rapid Feedback
- Software is Your Primary Goal
- Travel Light

Supplementary Principles of AM
- Content Is More Important Than Representation
- Open and Honest Communication

Practices

Core Practices of AM
- Active Stakeholder Participation
- Apply The Right Artifact(s)
- Collective Ownership
- Create Several Models in Parallel
- Create Simple Content
- Depict Models Simply
- Display Models Publicly
- Iterate To Another Artifact
- Model in Small Increments
- Model With Others
- Prove it With Code
- Single Source Information
- Use the Simplest Tools

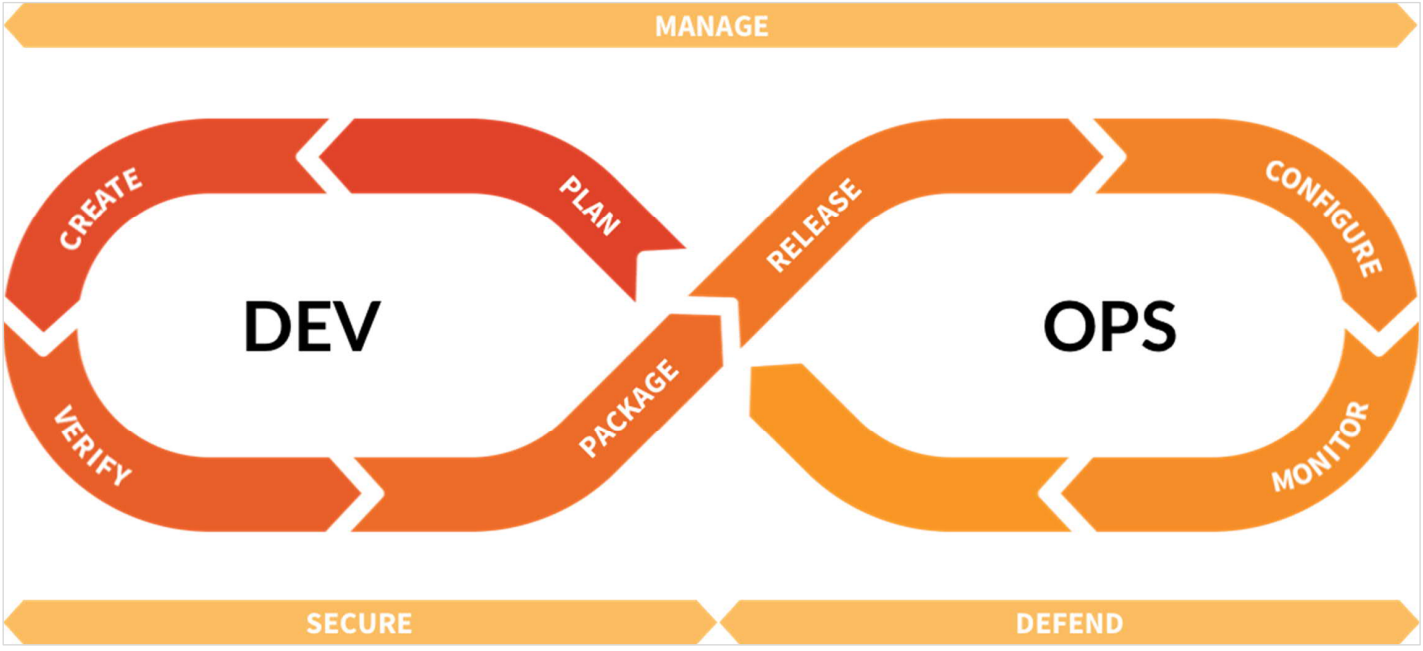
Supplementary Practices of AM
- Apply Modeling Standards
- Apply Patterns Gently
- Discard Temporary Models
- Formalize Contract Models
- Update Only When It Hurts

Tampereen yliopisto
Tampere University

DevOps

Tampereen yliopisto
Tampere University

DevOps loop



[https://about.gitlab.com/stages-devops-lifecycle/]

Tampereen yliopisto
Tampere University

DevOps values

DevOps is unique to software development methodology because its practice promotes empathy via a culture of collaboration and communication, rather than encouraging siloed functionality.

Culture: DevOps seeks to solve business problems that arise when people create and manage complex systems. In this regard, DevOps is as much a method for managing human problems as it is a technological solution. A "people over process over tools" culture is a central tenet of DevOps. Even with the advent of more innovative tools and advanced computing technology, the process of software development depends on elements of human culture

Automation: DevOps is not just about tools or about automating tasks using software. That said, automation is a core DevOps value, and it is essential to leveraging Agile development practices, including continuous integration and continuous delivery. To accommodate continuous releases, DevOps encourages automation. In the DevOps methodology, it's critical to prioritize problem solving that uses automation, and to make QA everyone's responsibility.

Measurement: In order to determine if DevOps is continuously improving processes, team members should collect and analyze data. This mandate applies to business-side metrics as well as development, test, and operations metrics.

Sharing: this value is referred as the loopback in the DevOps cycle, where stakeholders share ideas and solve problems. Sharing ideas helps attract talented people who thrive on feedback to improve. DevOps depends on the principles of continuous improvement and the collaboration those principles foster. Sharing is a core value of DevOps

Tampereen yliopisto
Tampere University

DevOps

DevOps refers to the combination of software development (which includes software testing) and operations methodology (the values, principles, methods, and practices) to deliver applications and services. DevOps promotes frequent communication and ongoing, real-time collaboration between traditionally disparate workflows of developers and IT operations teams. The DevOps approach to organizing workflows replaces siloed development and IT operations teams with multidisciplinary teams capable of implementing Agile planning and practices, such as continuous integration, continuous delivery, and infrastructure automation.

The technology community uses a variety of terms to describe the essence of DevOps. It is a culture, a movement, a philosophy, a set of practices, and the act of using tools (software) to automate and improve imperfect methods of managing complex systems.

The DevOps culture is reinforced by the practices it borrows from Agile and Lean principles, with an added focus on service and quality. By designing, building, testing, deploying, managing, and operating applications and systems faster and more reliably, DevOps practitioners seek to create value for customers (a profitable competitive advantage) and foster a manageable workflow that places people over product.

[www.smartsheet.com/devops]

DevOps practices

The DevOps methodology leverages key practices and techniques to **streamline software development and operations processes**. **Increasing the frequency and number of daily software deployments** is challenging to teams of all sizes with varying access to resources. To manage the organizational challenges of DevOps, its practitioners leverage practices known as continuous integration (CI) and continuous delivery (CD).

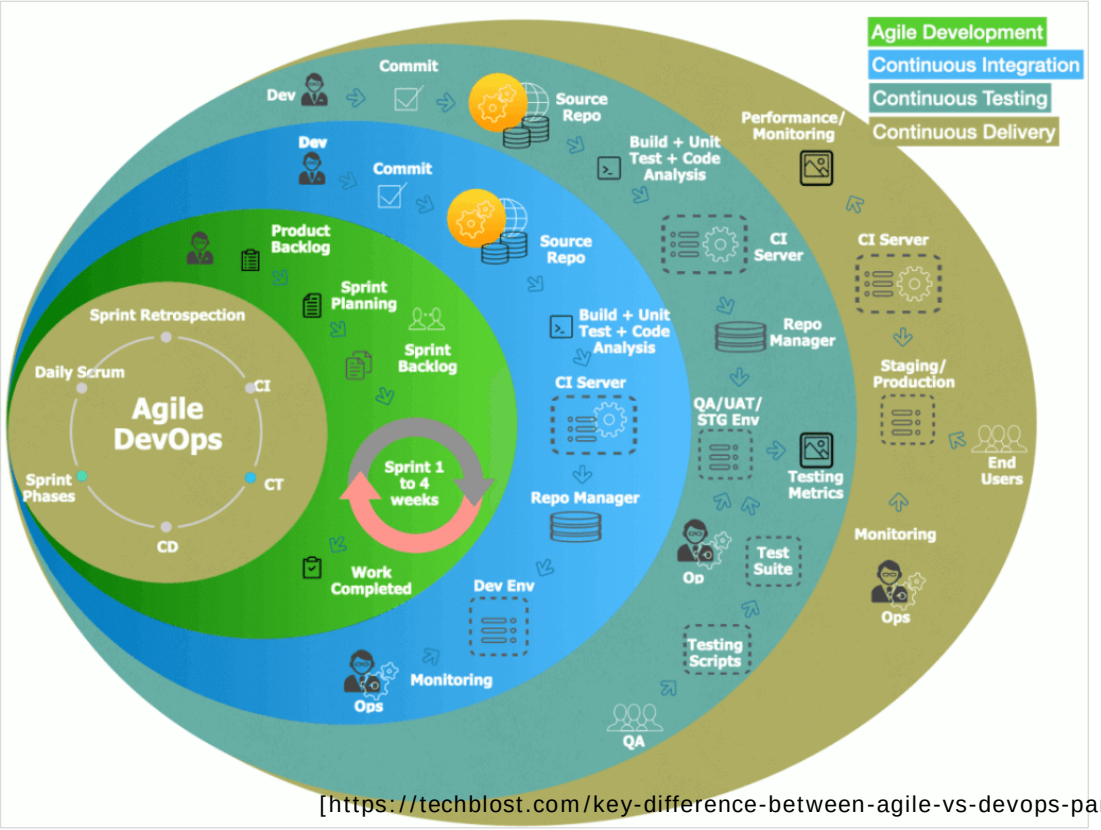
• **Continuous Integration:** CI is the practice of engaging in ongoing testing (leveraging automation) by **merging code development with real-time, problem-seeking tests**. The goals of CI are to **reduce integration problems**, improve quality, reduce time to release, and empower feedback loops that contribute to higher-velocity (daily) deployments. CI leverages comprehensive, automated testing frameworks and continuously addresses problems to keep the system in a working state.

• **Continuous Delivery:** CD is the practice of **frequently building, testing, and releasing code changes** to an (production or testing) environment in small batches after the build stage. The emphasis on **automated testing** (and **automated builds**) for quality assurance capitalizes on the efficiency of successful test automation and is essential to the deployment-ready goal of this practice. You can achieve CD when you synchronize it with the steps required for CI. However, CD does not require the deployment of every build. Continuous deployment deploys every CI build to production.

[https://www.smartsheet.com/devops]

Just another view...

Agile DevOps



[https://techblost.com/key-difference-between-agile-vs-devops-part-5/]

10 best practices for DevOps (2015)

[https://www.techrepublic.com/blog/10-things/10-best-practices-for-devops/]

DevOps - a methodology that can speed app development and delivery by getting application developers and operations specialists to collaborate throughout the end-to-end app development and deployment process.

1: Break the silos in IT

2: Adjust performance reviews

3: Create real-time project visibility

4: Use software automation wherever you can

5: Choose tools that are compatible with each other

6: Start with projects that are small and ensured for success

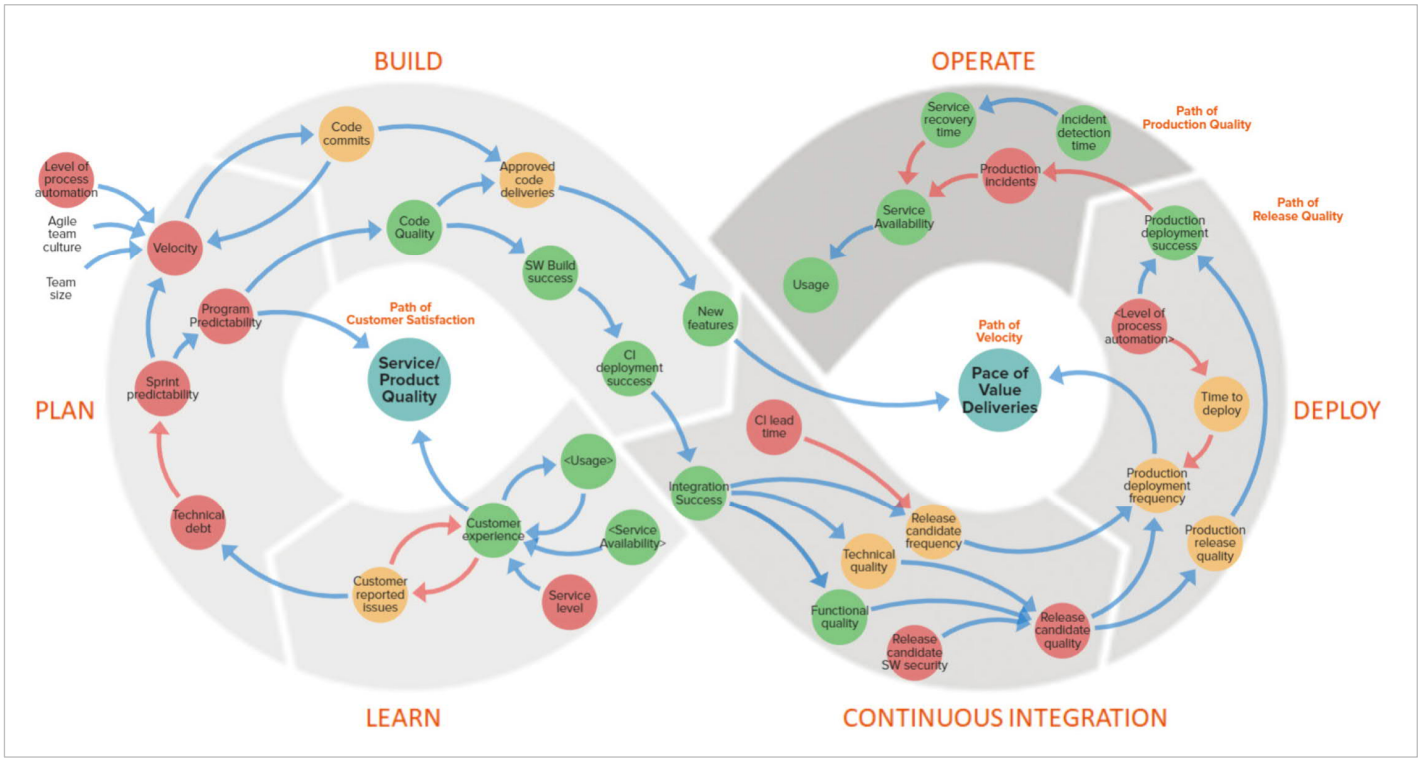
7: Don't forget the users!

8: Collaboratively manage change

9: Continuously deploy applications

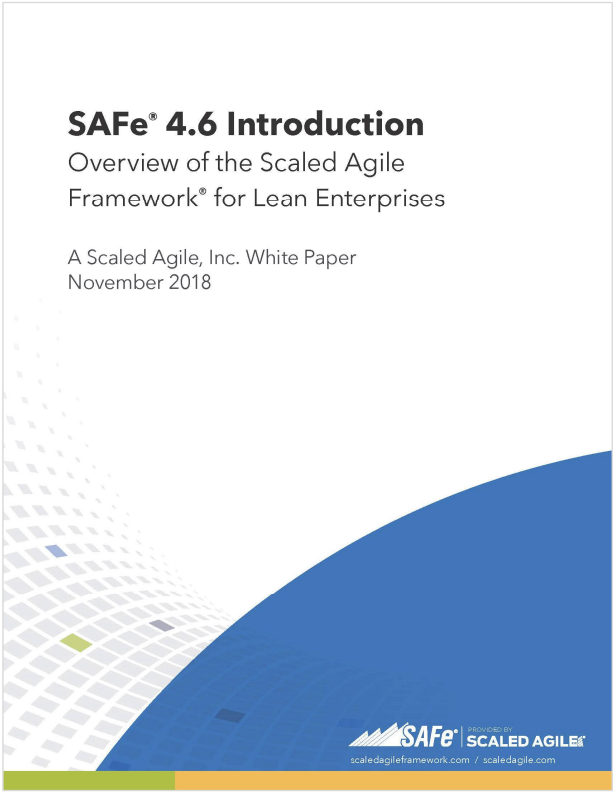
10: Create a service environment within the company.

[https://info.qentinel.com/hubfs/DevOps%20Value%20Creation%20Model.png]



Scaled agile framework (SAFe)

SAFe = Scaled Agile Framework



SAFe is based on nine immutable, underlying Lean-Agile principles.

These tenets and economic concepts inspire and inform the roles and practices of SAFe, influencing leadership behaviors and decision-making.

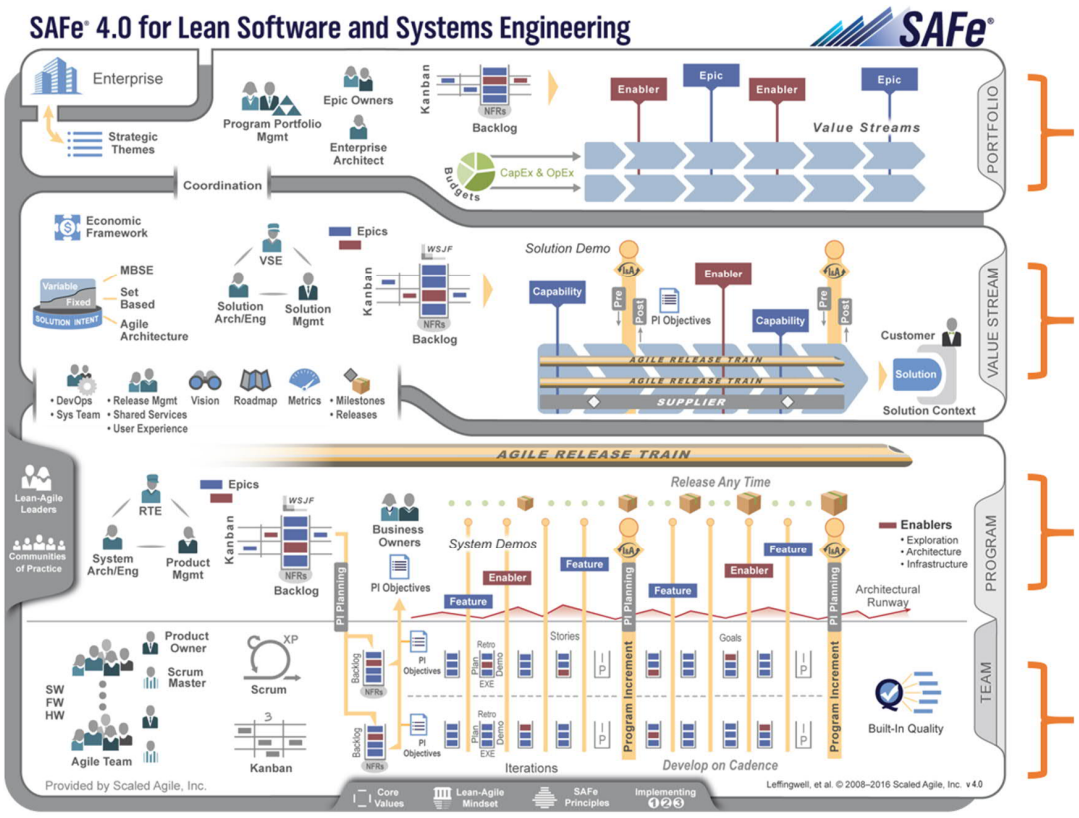
1. Take an economic view - An understanding of economics drives decisions.
2. Apply systems thinking - Everyone understands and commits to the common goals of the larger system.
3. Assume variability; preserve options - Decisions are delayed until the last responsible moment.
4. Build incrementally with fast, integrated learning cycles - Cadence-based learning cycles are used to gain knowledge, evaluate alternatives and inform decision-making.
5. Base milestones on objective evaluation of working systems - Progress is measured by objectives measures.
6. Visualize and limit WIP, reduce batch sizes, and manage queue length - Small batches of work, controlled Work in Progress (WIP), and small queues ensures fast flow of value and learning.
7. Apply cadence; synchronize with cross-domain planning - Regular synchronization continually aligns all system builders and ensure all perspectives are understood and resolved.
8. Unlock the intrinsic motivation of knowledge workers - Knowledge workers exhibit curiosity and have fundamentally different motivations.
9 . Decentralize decision-making - Autonomy empowers individuals and enhances motivation.

[www.scaledagile.com/resources/safe-whitepaper/]

Scaled agile framework.

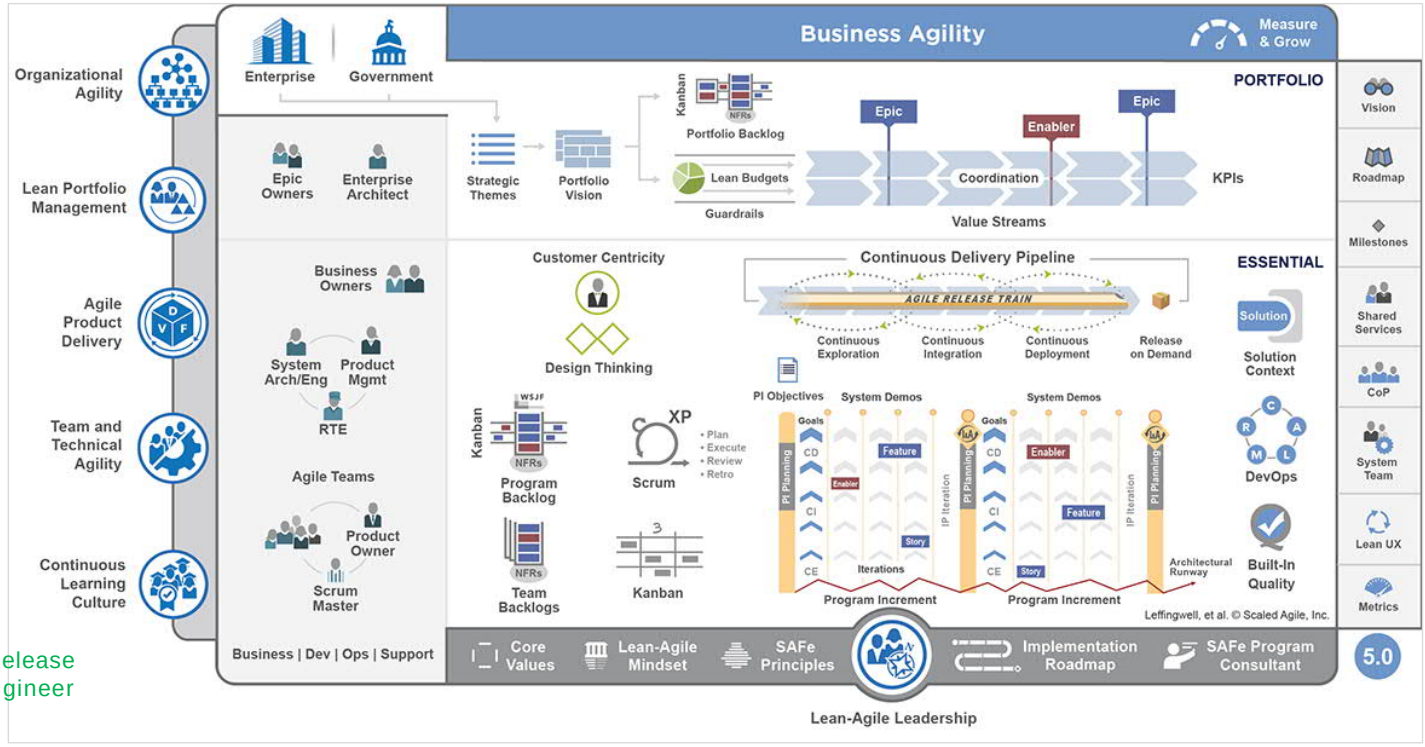
Four lines.

RTE = release train engineer



[www.scaledagileframework.com]

Scaled agile framework 5.0



RTE = release train engineer

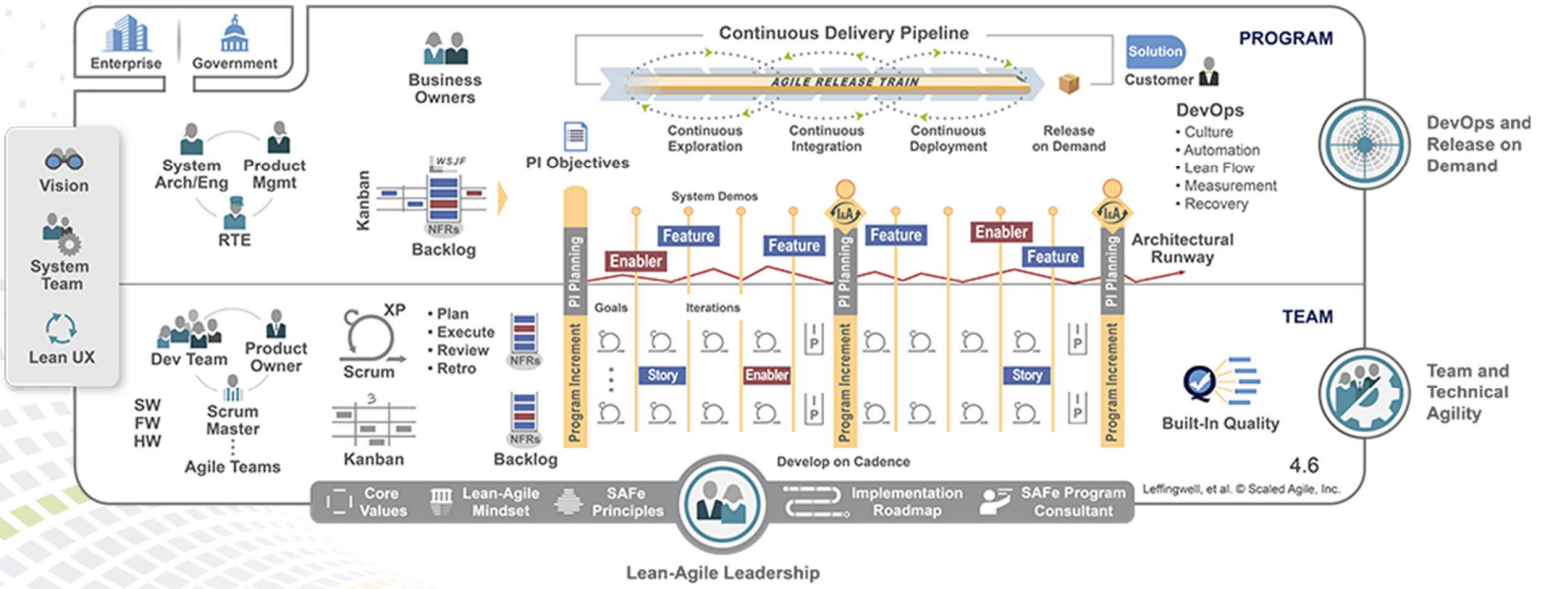
Essential SAFe

The Essential SAFe configuration is the most basic configuration of SAFe. It provides a starting point for implementing SAFe and describes the most critical elements needed to realize the majority of the framework's benefits. It consists of the Team and Program levels, and Foundation.

The Ten Essential Elements

1. Lean-Agile Principles
2. Real Agile Teams and Trains
3. Cadence and synchronization
4. PI planning (PI = program increment)
5. DevOps and releasability
6. System demo
7. Inspect and adapt
8. IP iteration
9. Architectural runway
10. Lean-Agile leadership.

Essential SAFe®



Kanban

Tampereen yliopisto
Tampere University

kanban

Kanban term came into existence using the flavors of “visual card,” “signboard,” or “billboard”, “signaling system” to indicate a workflow that limits Work In Progress (WIP). Kanban cards were originally used in Toyota to limit the amount of inventory tied up in “work in progress” on a manufacturing floor. Kanban not only reduces excess inventory waste, but also the time spent in producing it. In addition, all of the resources and time freed by the implementation of a Kanban system can be used for future expansions or new opportunities.

The core concept of Kanban includes

- Visualize Workflow
  - Split the entire work into defined segments or states, visualized as named columns (lists) on a wall (board).
  - Write each item on a card and put in a column to indicate where the item is in the workflow.
- Limit WIP
  - Assign explicit limits to how many items can be in progress at each workflow segment / state. i.e., Work in Progress (WIP) is limited in each workflow state.
- Measure the Lead Time
  - Lead Time, also known as cycle time is the average time to complete one item. Measure the Lead Time and optimize the process to make the Lead Time as small and predictable as possible.

[https://www.tutorialspoint.com/kanban/]

Tampereen yliopisto
Tampere University

kanban

The Kanban Method is a means to design, manage, and improve flow systems for knowledge work. The method also allows organizations to start with their existing workflow and drive evolutionary change. They can do this by visualizing their flow of work, limit work in progress (WIP) and stop starting and start finishing.

The Kanban Method gets its name from the use of kanban – visual signaling mechanisms to control work in progress for intangible work products.

A general term for systems using the Kanban Method is flow – reflecting that work flows continuously through the system instead of being organized into distinct timeboxes.

Kanban can be used in any knowledge work setting, and is particularly applicable in situations where work arrives in an unpredictable fashion and/or when you want to deploy work as soon as it is ready, rather than waiting for other work items.

[www.agilealliance.org/glossary/]

https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/

The 4 Core Principles of Kanban

Kanban method is an approach to incremental, evolutionary process and systems change for knowledge work organizations. It is focused on getting things done and the most important principles can be broken down into four basic principles and six practices.

Principle 1: Start With What You Do Now

Principle 2: Agree to Pursue Incremental, Evolutionary Change

Principle 3: Respect the Current Process, Roles & Responsibilities

Principle 4: Encourage Acts of Leadership at All Levels.

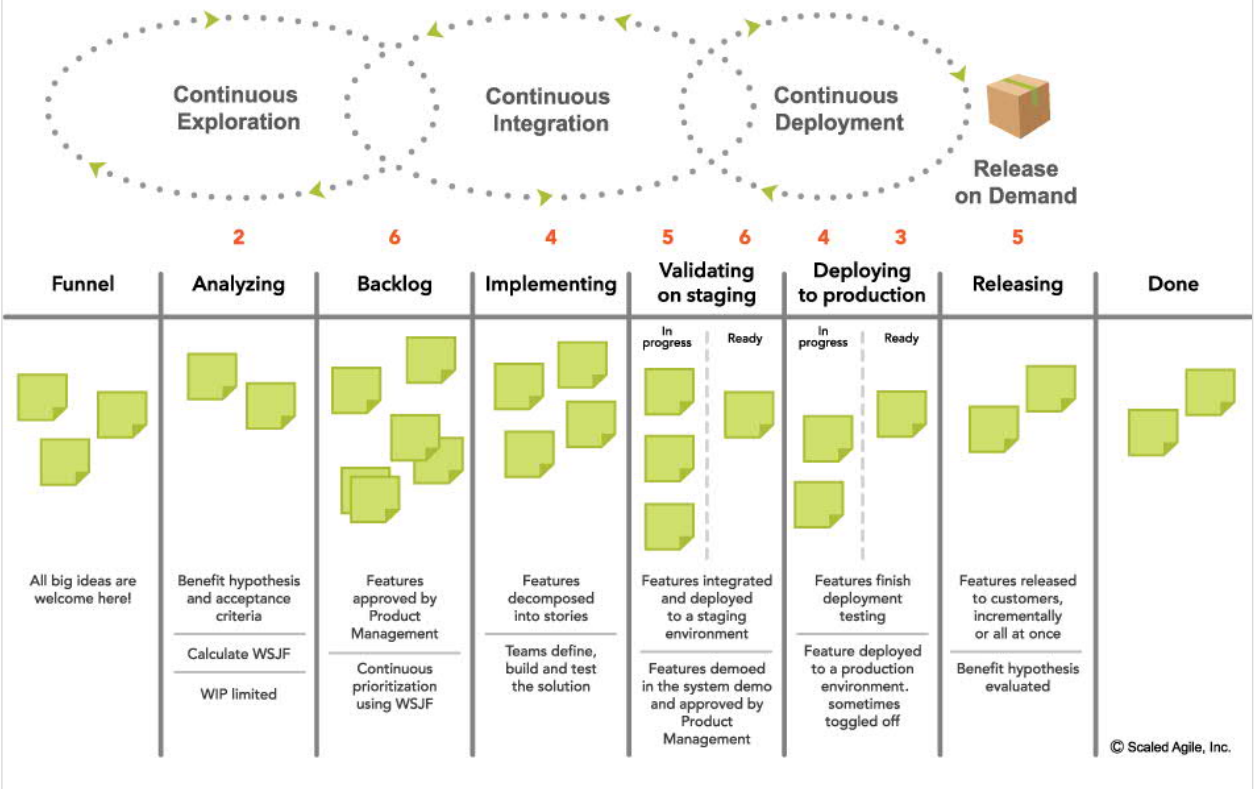
The 6 Practices of Kanban

- **Visualize the Workflow**
- **Limit Work in Progress**
- **Manage Flow**
- **Make Process Policies Explicit**
- **Feedback Loops**
- **Improve Collaboratively (using models & the scientific method).**

The program Kanban facilitates the flow of features through the continuous delivery pipeline.

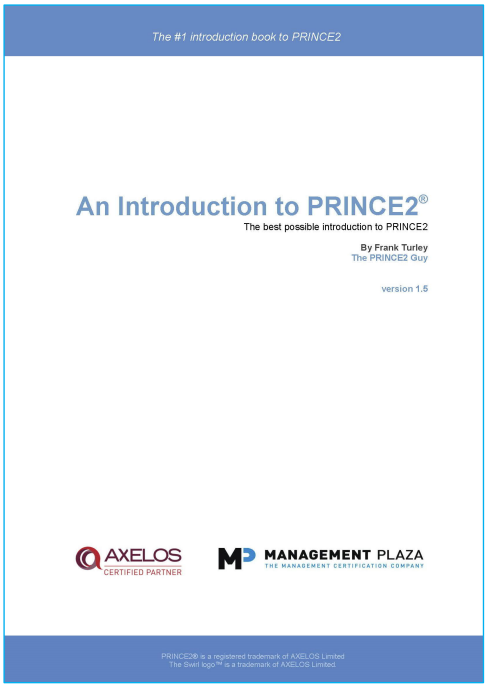
WSJF = weighted shortest job first

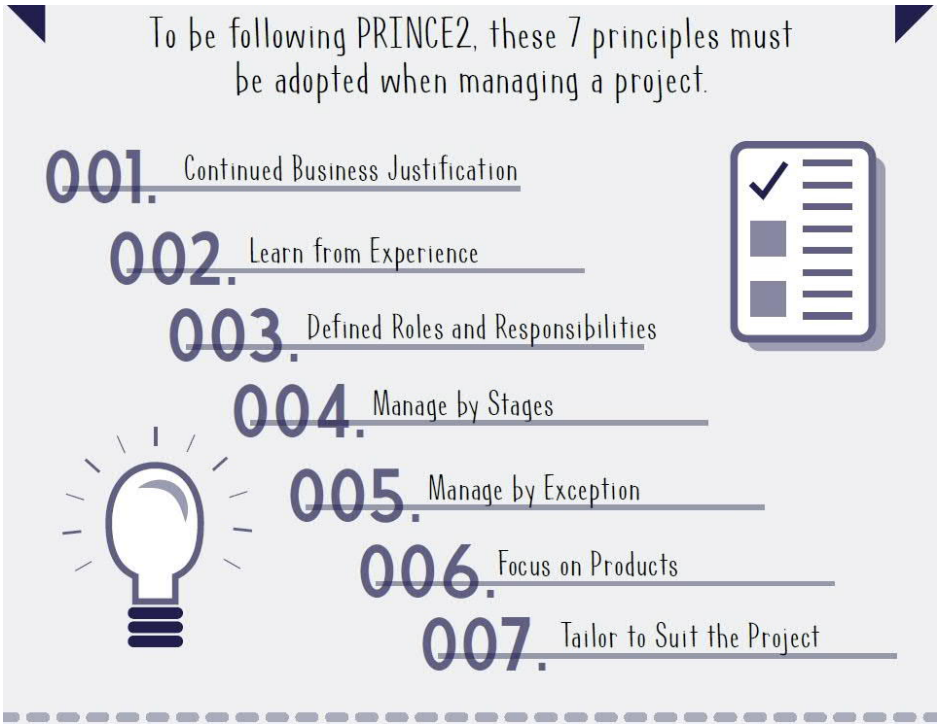
WIP = work in process



Projects in controlled environments (PRINCE)

PRINCE2, PRojects IN Controlled Environments v.2





[https://www.itonlinelearning.com/prince2-cheat-sheet/]

Tampereen yliopisto
Tampere University

PRINCE2

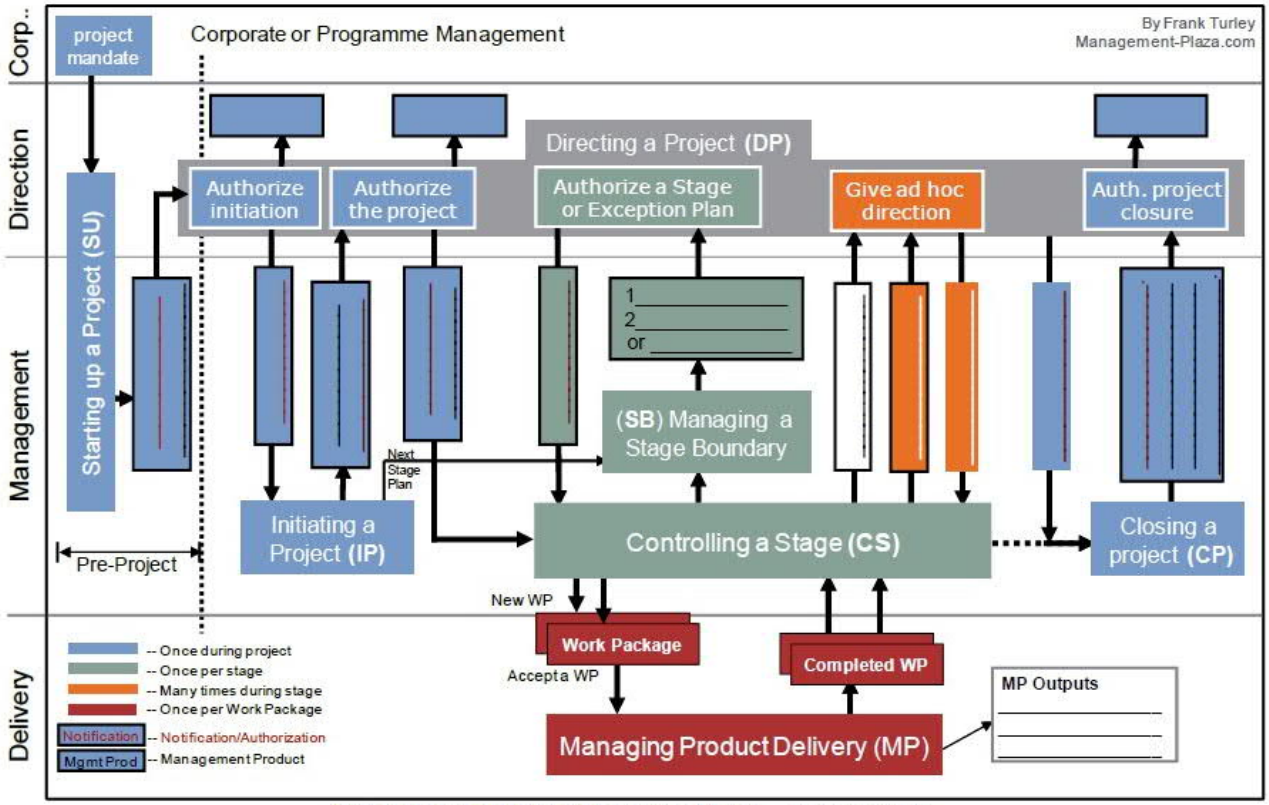


Fig 2.1 The PRINCE2 Process Model – 7 processes

Tampereen yliopisto
Tampere University

Scrumban

Tampereen yliopisto
Tampere University

Scrumban

SCRUMBAN CHEAT SHEET

PROCESS
Continuous work with planning on demand and detached releases once the product is ready.

ROLES
Scrumban does not have any predesigned roles, but focuses on managing a team of different specialties.

ITERATIONS
Scrumban does not have defined iterations, instead the team plans on the need basis and releases once they feel the product is ready.

TASK SIZE
There is no particular size a task should fit, instead it should have a clear goal for a team member to complete.

TASK ASSIGNMENT
Team members pull tasks from priority columns based on their skillset and priority.

NEW TASKS IN ITERATION

New tasks can be added to a running iteration.

ESTIMATION
Optional, measured against the average cycle time.

WORK IN PROGRESS
Limits the number of tasks the team can work on at any given point. Guarantees each task is completed before the next one is started.

SCOPE LIMITS
Work In Progress (WIP) limits the current work amount.

TERMS

Average Cycle Time
Calculates the average time from task being started (not created) to it being delivered.

Triage Trigger
Notifies the team there is only 5-15% of the overall time left until the release date.

Triage
An event where the team decides which tasks will be completed and which won't.

Feature Freeze
Done after Triage, the team stops all planning and adding any new features.

Planning Buckets
Used to introduce long-term planning – includes 1 year, 6 and 3 months options.

Planning Trigger
Determines how many tasks should be left in the backlog before planning session occurs.

MEETINGS
All Scrumban meetings are held once the team needs them and do not occur on a planned schedule.
Planning Session – set off by a planning trigger is for filling the backlog and prioritizing tasks.
Daily Standup – a 15 minute standup where team members present what they have done and will do.
Kaizen – similar to retrospective, the team discusses the release and determines what can be improved.

PRIORITIZATION


Recommended on each planning

BOARD


Usually combined out of 3 sections:
1. To Do for planned and prioritized tasks.
2. Doing for tasks the team is working on.
3. Done for completed tasks.

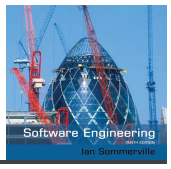
Brought To You By: eylean
www.eylean.com

Tampereen yliopisto
Tampere University

Highlights - What to remember

- there are many many methods, no one fits all sw dev projects
- ”Aim for best practices and standard conventions.”
- agile means taking customer’s changes into account all the time during the project
- agile documentation is minimum, for user’s and maintenance needs; code comments, Product Backlog, some Test log/report evidence, perhaps User Manual, Maintenance quide,...

Key points



- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, reducing process overheads by minimizing documentation and producing high-quality code.
- Agile development practices include
  - User stories for system specification
  - Frequent releases of the software,
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team.

Key points

- ✧ Scrum is an agile method that provides a project management framework.
  - It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- ✧ Many practical development methods are a mixture of plan-based and agile development.
- ✧ Scaling agile methods for large systems is difficult.
  - Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.

Tampereen yliopisto
Tampere University

Now the additional L7 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

Now the additional L7 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

Now the additional L7 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

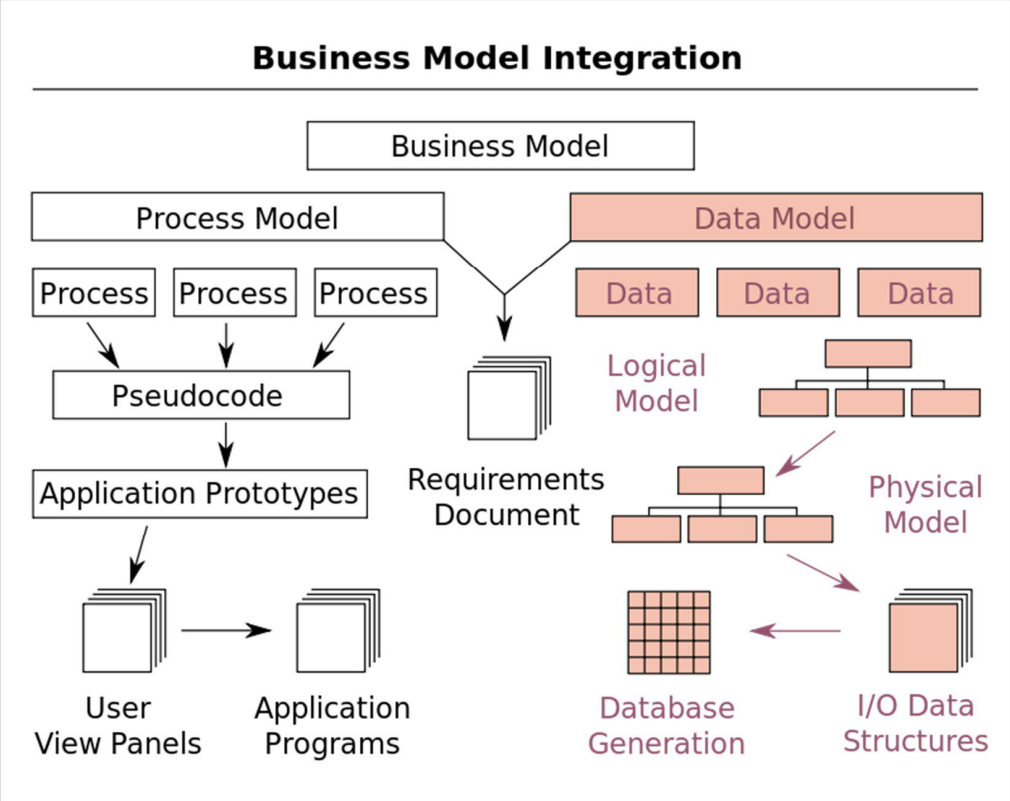
Tampereen yliopisto
Tampere University

Now the additional L7 extra slides are here

No time to show these at lectures, but otherwise good to know, at least if you are a major reader.

Tampereen yliopisto
Tampere University

Software development life-cycle



[Paul R. Smith. Redrawn by Marcel Douwe Dekker]

Tampereen yliopisto
Tampere University

Waterfall

Tampereen yliopisto
Tampere University

One early (1950..1960) way of describing program development

The first mentioning of waterfall model:
Herbert D. Benington, Symposium on advanced programming methods for digital computers, 1956.

The first formal description:
Winston W. Royce, "Managing the Development of Large Software Systems", 1970.

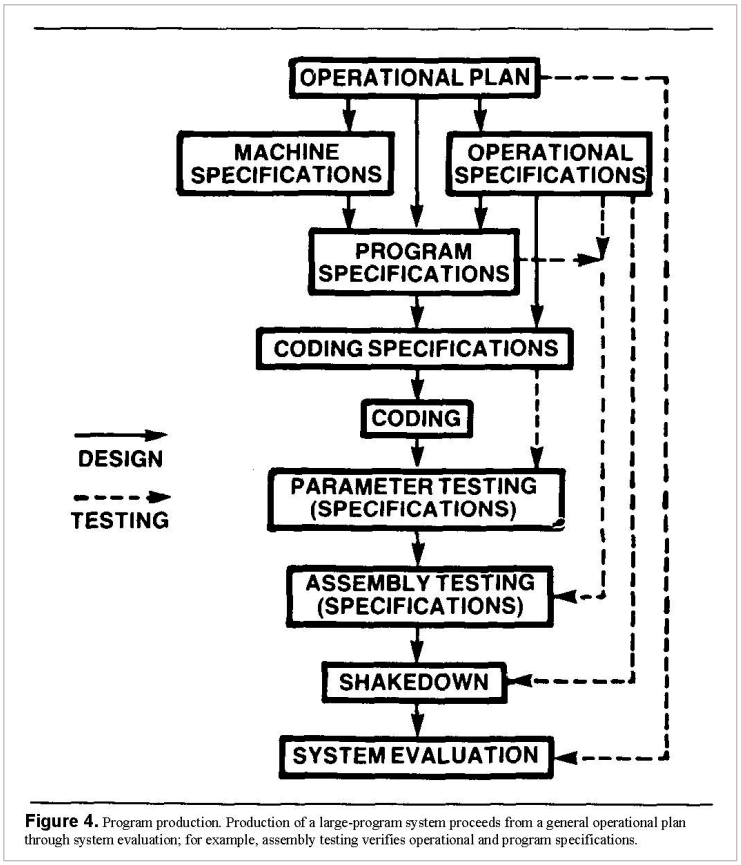


Figure 4. Program production. Production of a large-program system proceeds from a general operational plan through system evaluation; for example, assembly testing verifies operational and program specifications.

Why methods;

Software systems became built more and more, so defining some solid way of working made life a bit easier.

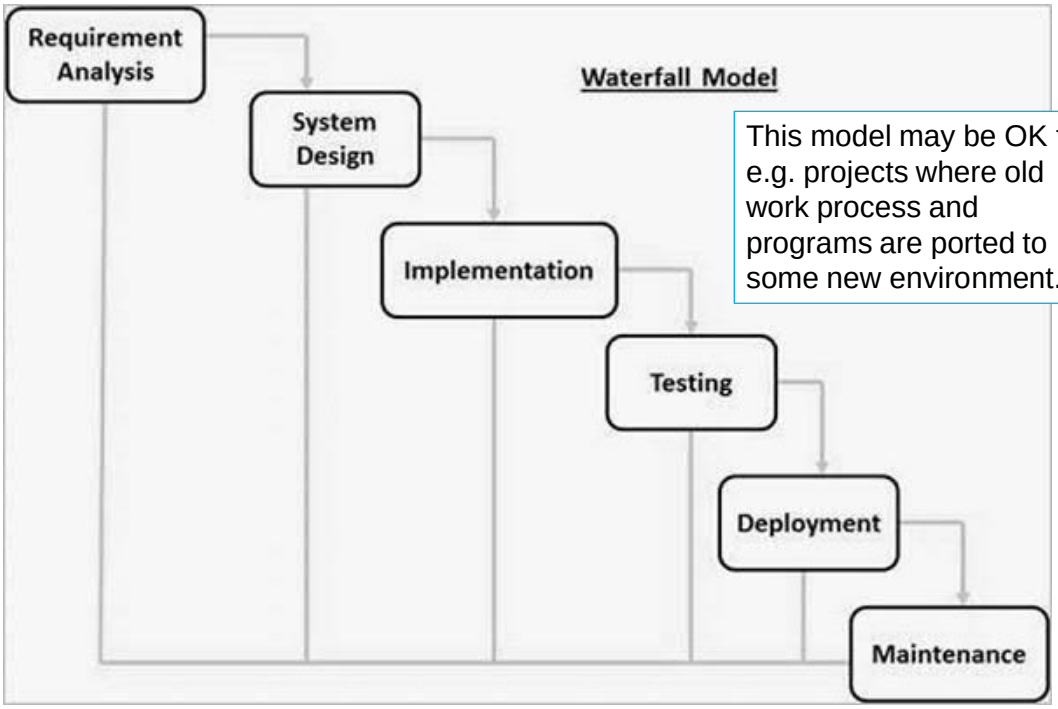
Tampereen yliopisto
Tampere University

waterfall model (the oldest)

The waterfall model was the first one, it show the separate phases of SDLC.

However, it was ”fixed”; after one phase you did not iterate. So you have to do it ”right at the first try”, which usually do not happen.

At the end, after the whole project is ready, customer sees the result (deployment).



This model may be OK for e.g. projects where old work process and programs are ported to some new environment.

[https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm]

Kaisti *et al. EURASIP Journal on Embedded Systems* 2013, **2013**:15
http://jes.eurasipjournals.com/content/2013/1/15

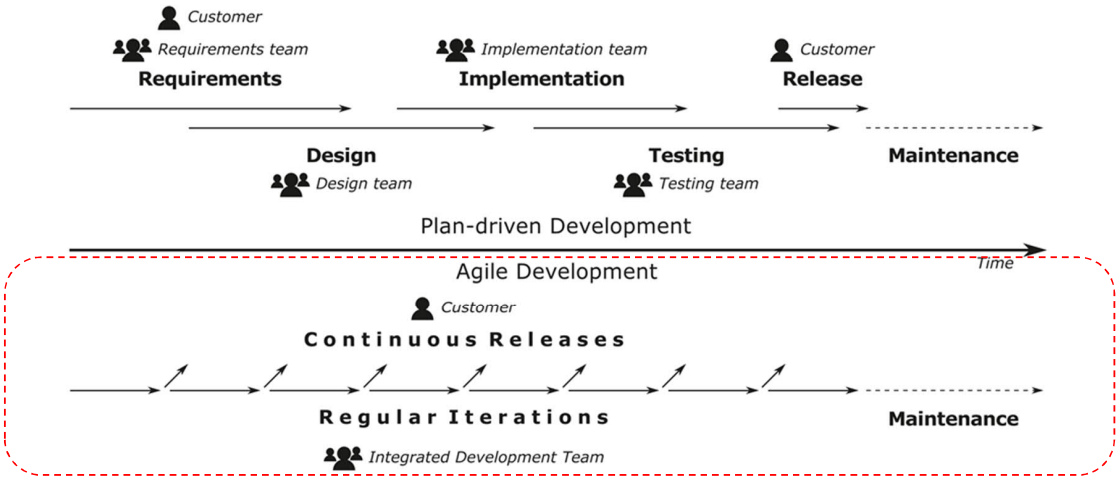


Figure 1 The main differences between the plan-driven and the agile development methodologies.

Tampereen yliopisto
Tampere University

Spiral

Tampereen yliopisto
Tampere University

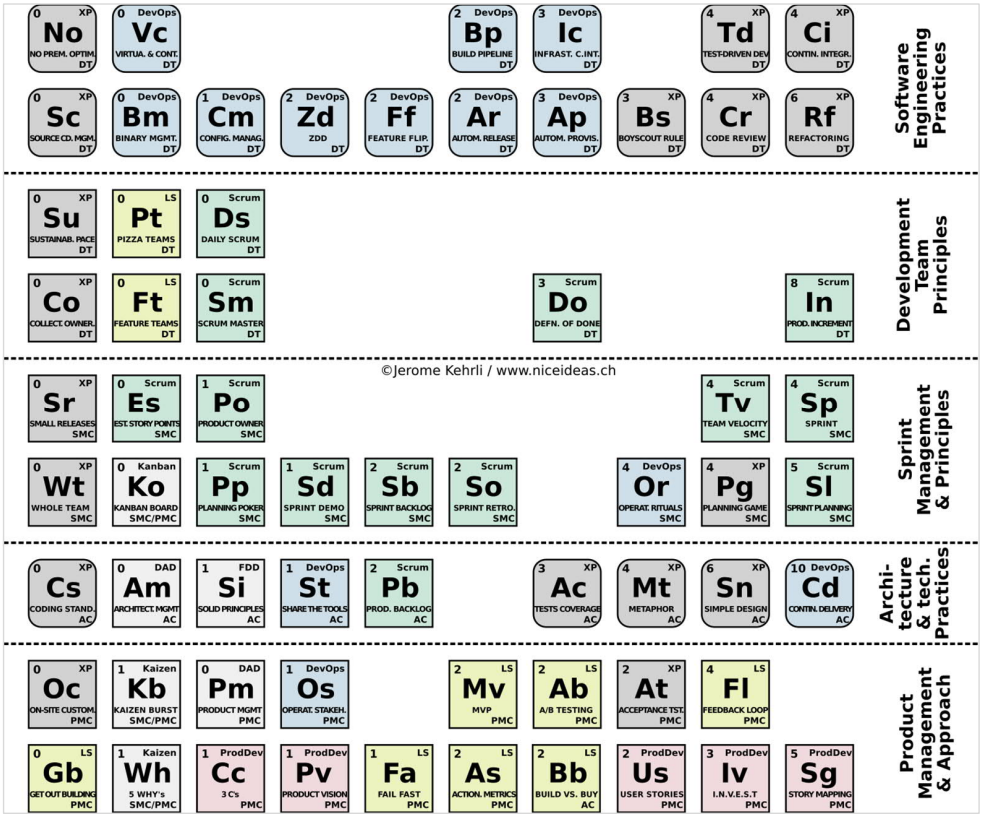
Iterative
Incremental

Tampereen yliopisto
Tampere University

Agile

Just another...

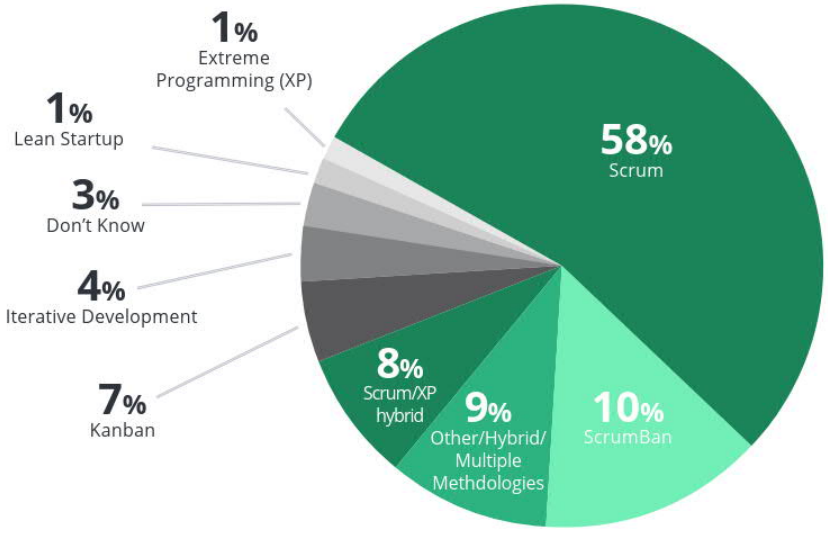
Periodic Table of Agile Principles and Practices



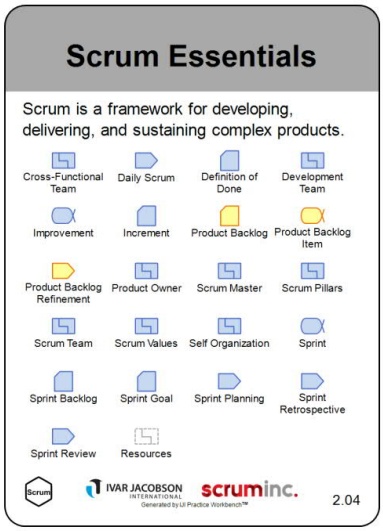
AGILE METHODS AND PRACTICES

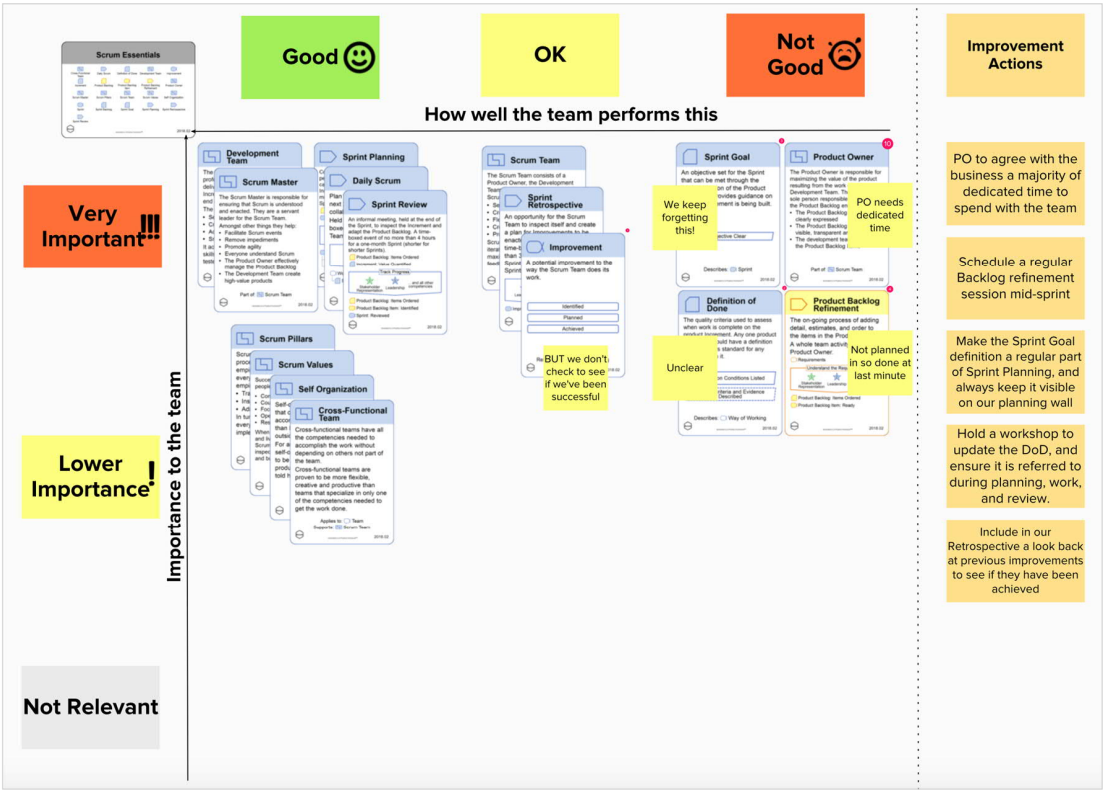
AGILE METHODOLOGIES USED

Scrum and related variants continue to be the most common Agile methodologies used by respondents' organizations.



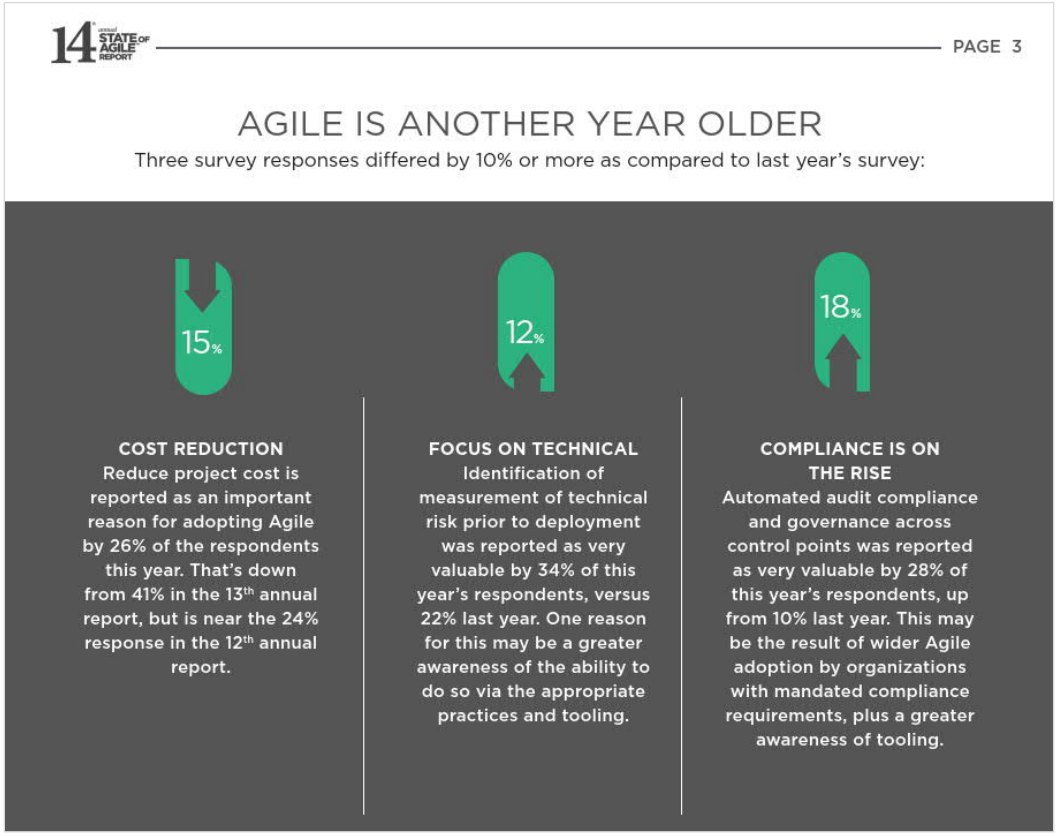
Total exceeds 100% due to rounding.


Scrum Essentials
Scrum is a framework for developing, delivering, and sustaining complex products.
Cross-Functional Team
Daily Scrum
Definition of Done
Development Team
Improvement
Increment
Product Backlog
Product Backlog Item
Product Backlog Refinement
Product Owner
Scrum Master
Scrum Pillars
Scrum Team
Scrum Values
Self Organization
Sprint
Sprint Backlog
Sprint Goal
Sprint Planning
Sprint Retrospective
Sprint Review
Resources
IVAR JACOBSON INTERNATIONAL
scruminc.
2.04


Good
OK
Not Good
Improvement Actions
How well the team performs this
Very Important!!!
Lower Importance!
Not Relevant
Importance to the team
Development Team
Scrum Master
Sprint Planning
Daily Scrum
Sprint Review
Scrum Team
Sprint Retrospective
Improvement
BUT we don't check to see if we've been successful
Sprint Goal
Product Owner
We keep forgetting this!
PO needs dedicated time
Definition of Done
Unclear
Product Backlog Refinement
Not planned in so done at last minute
Scrum Pillars
Scrum Values
Self Organization
Cross-Functional Team
PO to agree with the business a majority of dedicated time to spend with the team
Schedule a regular Backlog refinement session mid-sprint
Make the Sprint Goal definition a regular part of Sprint Planning, and always keep it visible on our planning wall
Hold a workshop to update the DoD, and ensure it is referred to during planning, work, and review.
Include in our Retrospective a look back at previous improvements to see if they have been achieved

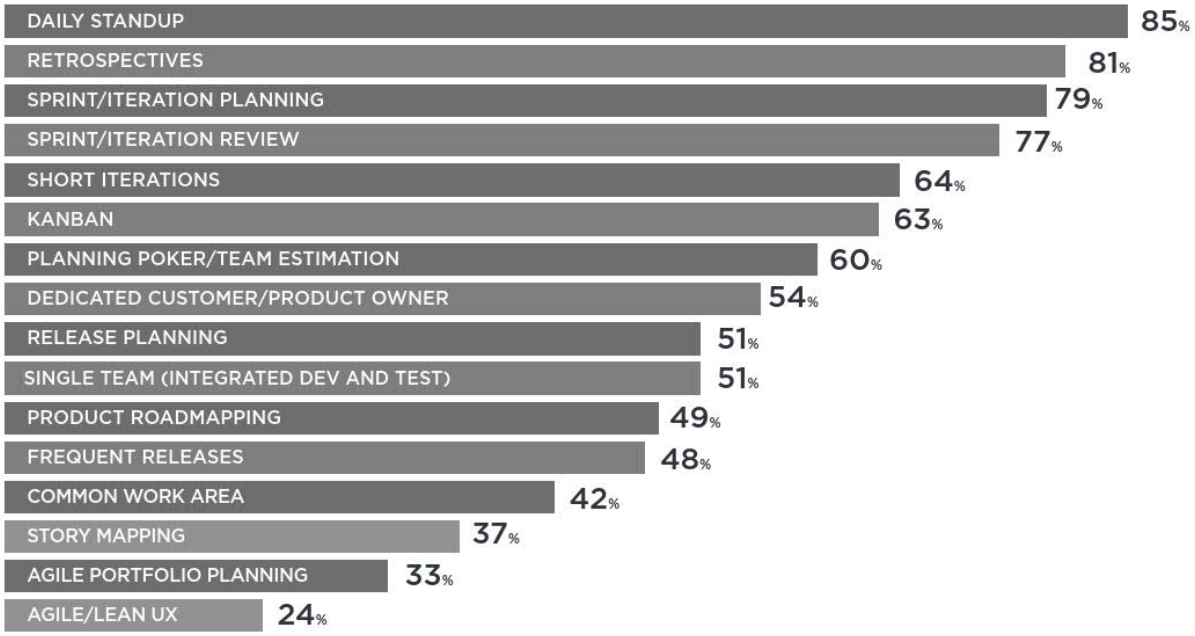
2020


14th annual STATE OF AGILE REPORT
digital.ai


14 annual STATE OF AGILE REPORT
PAGE 3
AGILE IS ANOTHER YEAR OLDER
Three survey responses differed by 10% or more as compared to last year's survey:
15%
12%
18%
COST REDUCTION
Reduce project cost is reported as an important reason for adopting Agile by 26% of the respondents this year. That's down from 41% in the 13th annual report, but is near the 24% response in the 12th annual report.
FOCUS ON TECHNICAL
Identification of measurement of technical risk prior to deployment was reported as very valuable by 34% of this year's respondents, versus 22% last year. One reason for this may be a greater awareness of the ability to do so via the appropriate practices and tooling.
COMPLIANCE IS ON THE RISE
Automated audit compliance and governance across control points was reported as very valuable by 28% of this year's respondents, up from 10% last year. This may be the result of wider Agile adoption by organizations with mandated compliance requirements, plus a greater awareness of tooling.

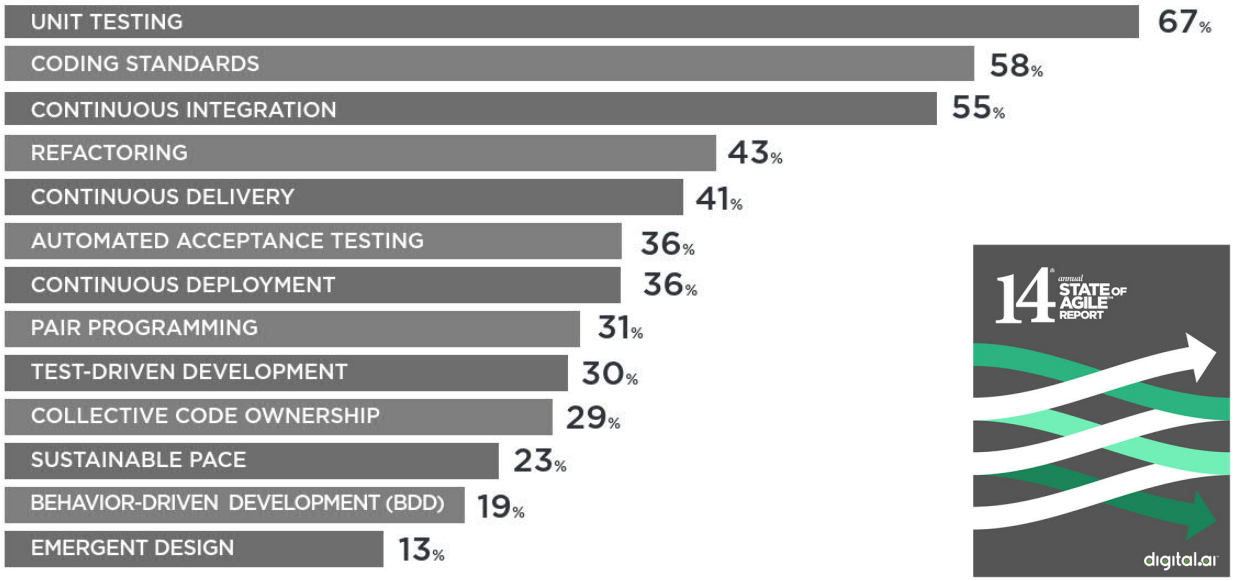
AGILE TECHNIQUES EMPLOYED

Notable changes in Agile techniques and practices that respondents said their organization uses were an increase in product roadmapping (49% this year compared to 45% last year) and a decrease in release planning (51% this year compared to 57% last year).



ENGINEERING PRACTICES EMPLOYED

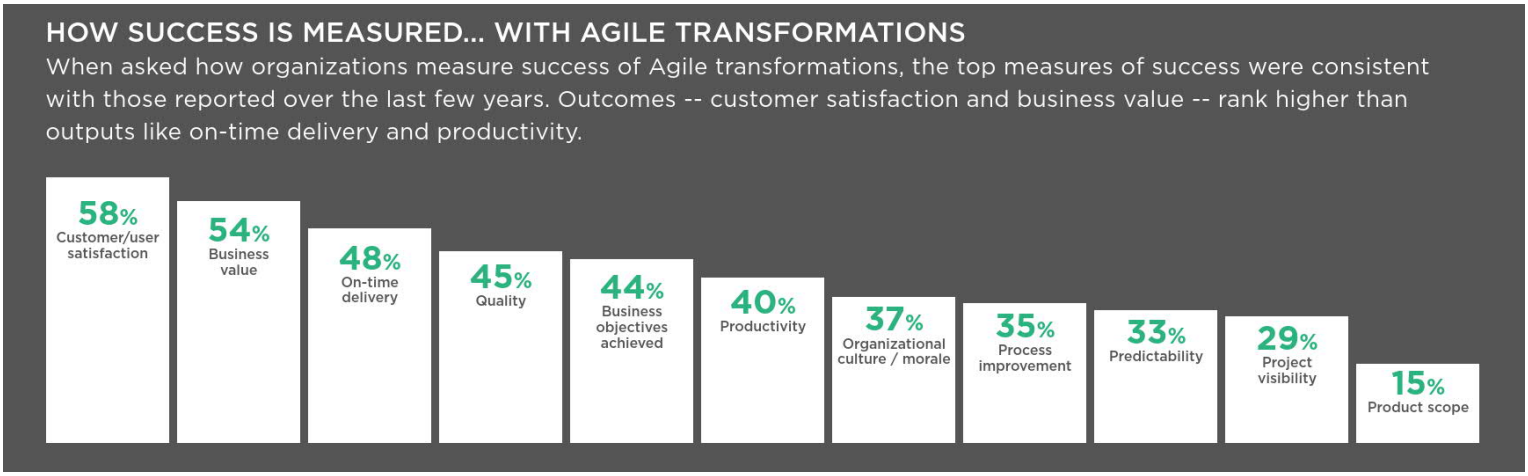
The overall rank order of engineering practices employed remained almost the same this year over last. Automated acceptance testing increased 3% while pair programming, test-driven development, and behavior-driven development each fell 3%.



*Respondents were able to make multiple selections

This is how responders measure software development success.

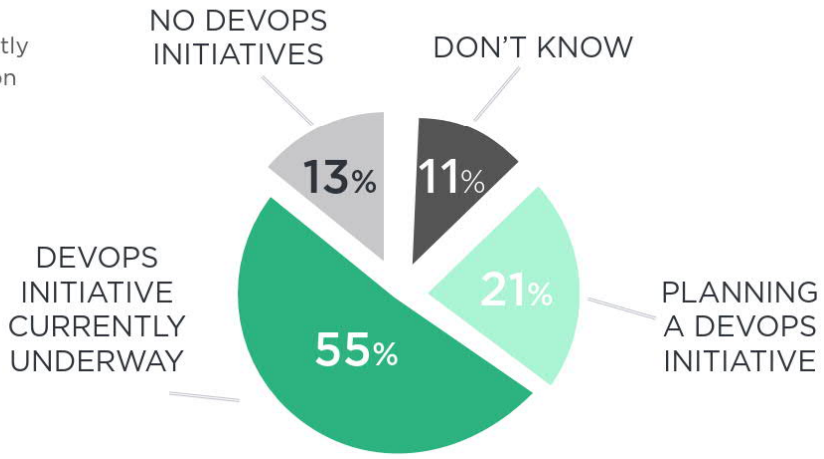
Not what was the success rates.

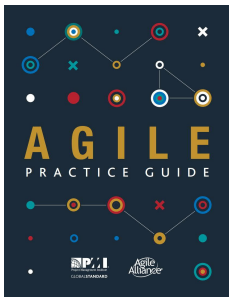


DEVOPS & VALUE STREAM MANAGEMENT

DEVOPS INITIATIVES

76% of respondents stated that they currently have a DevOps initiative in their organization or are planning one in the next 12 months (compared to 73% last year).





Iteration-Based Agile

Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test	Repeat as needed ...	Requirements Analysis Design Build Test	Requirements Analysis Design Build Test

NOTE: Each timebox is the same size. Each timebox results in working tested features.

Flow-Based Agile

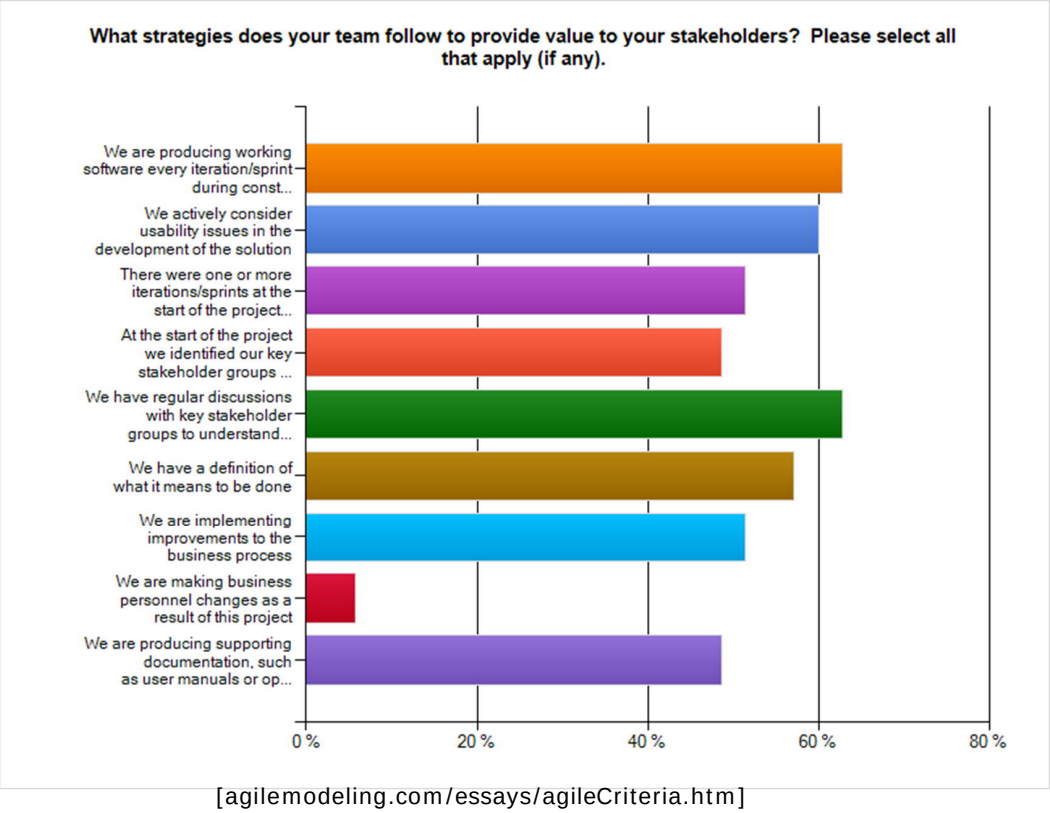
Requirements Analysis Design Build Test the number of features in the WIP limit	Requirements Analysis Design Build Test the number of features in the WIP limit	Requirements Analysis Design Build Test the number of features in the WIP limit	Repeat as needed ...	Requirements Analysis Design Build Test the number of features in the WIP limit	Requirements Analysis Design Build Test the number of features in the WIP limit

NOTE: In flow, the time it takes to complete a feature is not the same for each feature.

Figure 3-5. Iteration-Based and Flow-Based Agile Life Cycles

Tampereen yliopisto
Tampere University

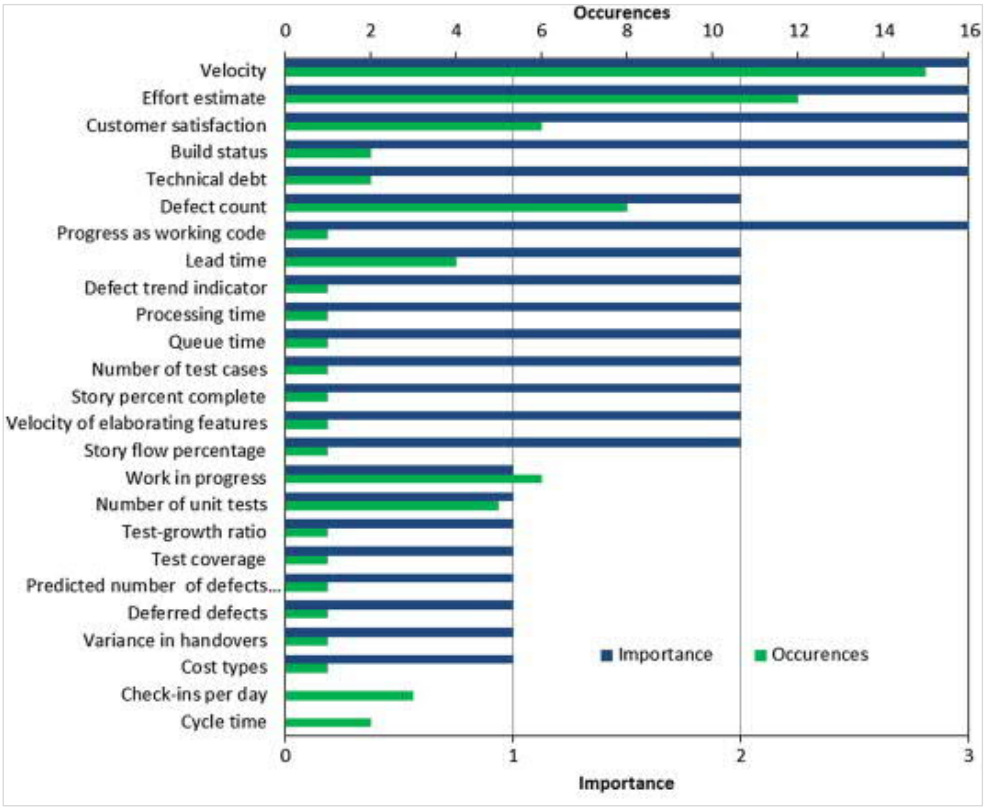
Agile criteria survey



[agilemodeling.com/essays/agileCriteria.htm]

Tampereen yliopisto
Tampere University

Agile metrics survey



[Using metrics in Agile and Lean Software Development, 2015]

Table A.1 Typical agile methods

Method	Source and date	Description
Crystal Clear	Alistair Cockburn, early 2000s	Puts particular emphasis on the interactions with the team through "osmotic communication." With osmotic communication, questions and answers flow naturally and with surprisingly little disturbance among team members.
Disciplined agile delivery (DAD)	Scott Ambler and Mark Lines, 2010s	Combines elements from Scrum, Lean Software Development, and other agile methodologies to support incremental and iterative software solutions on an enterprise scale.
Extreme Programming (XP)	Kent Beck, Ward Cunningham, Ron Jeffries, 1995+	The basic advantage of XP is that the whole process is visible and accountable. The developers will make concrete commitments about what they will accomplish, show concrete progress in the form of deployable software, and when a milestone is reached they will describe exactly what they did and how and why that differed from the plan. This allows business-oriented people to make their own business commitments with confidence, to take advantage of opportunities as they arise, and eliminate dead-ends quickly and cheaply.
Kanban	1940s for manufacturing Mid-2000s for software	Model for introducing change through incremental improvements with an emphasis on continual delivery while not overburdening the development team.
Lean Software Development	Mary and Tom Poppendieck, early 2000s	Lean Software Development owes much of its principles and practices to the Lean Enterprise movement and the practices of companies like Toyota. Lean Software Development focuses the team on delivering value to the customer and on the efficiency of the "Value Stream," the mechanisms that deliver value.
Scrum	Ken Schwaber and Jeff Sutherland, early 1990s, based on work of Hirotaka Takeuchi and Ikujiro Nonaka	Emphasizes empirical feedback, team self-management, and striving to build properly tested product increments within short iterations.
Rational Unified Process (RUP)	Rational Software Corporation, 1990s	RUP is an adaptable process framework focused on iterations. It divides the software development life cycle into iterative cycles, with each cycle working on a new version of the product. Each cycle is broken up into four phases: Inception, Elaboration, Construction, and Transition. Within each phase, the tasks are categorized into nine disciplines, six engineering disciplines (business modeling, requirements, analysis and design, implementation, test, deployment) and three supporting disciplines. RUP supports the use of UML (Unified Modeling Language) as its primary notation for requirements, architecture, and design. RUP emphasizes the adoption of various best practices in order to reduce the risks in software development.

https://www.agilebusiness.org/page/whatisdsdm

DSDM is an Agile method that focuses on the full project lifecycle, DSDM (formally known as Dynamic System Development Method) was created in 1994, after project managers using RAD (Rapid Application Development) sought more governance and discipline to this new iterative way of working.

DSDM's success is due to the philosophy "that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to the business." Supporting this philosophy with the eight principles allows teams to maintain focus and achieve project goals.

The eight Principles of DSDM:

- Focus on the business need
- Deliver on time
- Collaborate
- Never compromise quality
- Build incrementally from firm foundations
- Develop iteratively
- Communicate continuously and clearly
- Demonstrate control.

Planning poker (estimation poker)

TABLE 3.2 A Popular Story Point Range

Story Point	T-shirt Size	
0	Freebie, item has already been implemented	
1	XS	Extra small
2	S	Small
3	M	Medium
5	L	Large
8	XL	Extra-large
13	XXL	Double extra-large
20	XXXL	Huge

[Agile product management, 2010]

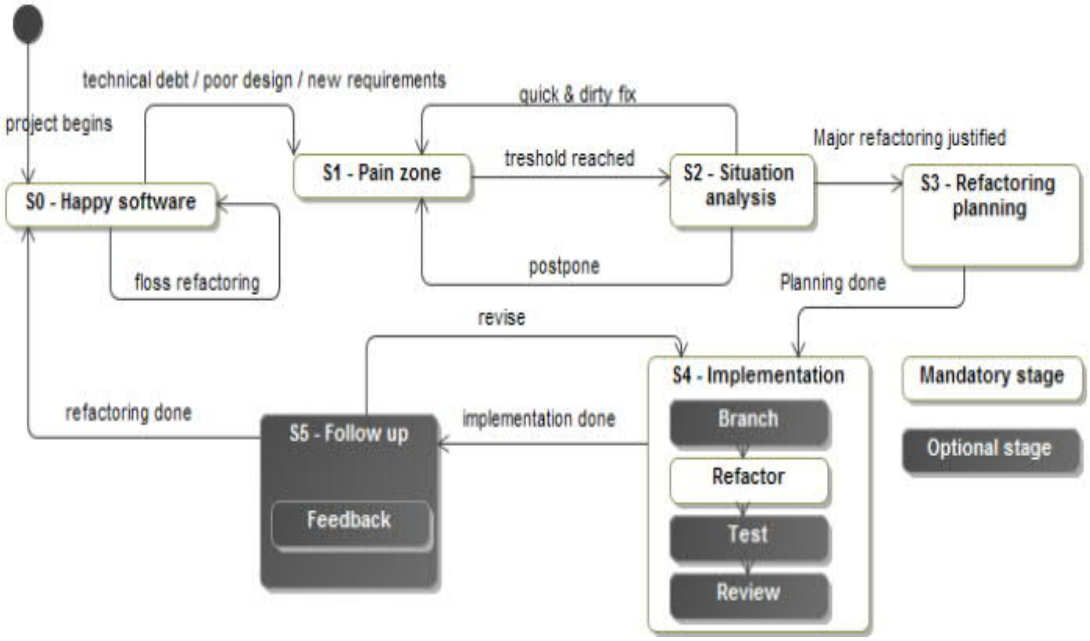
Tampereen yliopisto
Tampere University

Pair programming

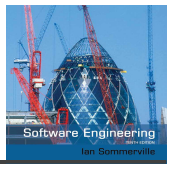
Tampereen yliopisto
Tampere University

Refactoring

Decision making framework for Refactoring[1]



Refactoring



- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Tampereen yliopisto
Tampere University

Definition of Done (DoD)

Tampereen yliopisto
Tampere University

Definition of done (DoD)

The team agrees on, and displays prominently somewhere in the team room, a list of criteria which must be met before a product increment "often a user story" is considered "done". Failure to meet these criteria at the end of a sprint normally implies that the work should not be counted toward that sprint's velocity.

Some teams use the term "Done List" or "Done Checklist".

- the Definition of Done limits the cost of rework once a feature has been accepted as "done"
- having an explicit contract limits the risk of misunderstanding and conflict between the development team and the customer or product owner.

DoD for a feature may be e.g. peer reviewed, unit tested (CI pipeline), checked against requirements, or merge/pull request (some other developer than author pushes code to version control).

Definition of Done, DoD

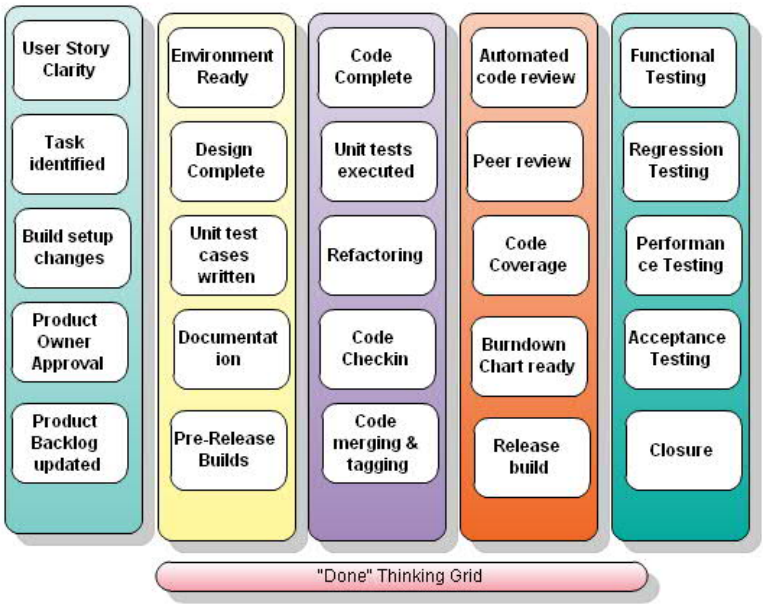
- The team agrees on, and displays prominently somewhere in the team room, a list of criteria which must be met before a product increment "often a user story" is considered "done". Failure to meet these criteria at the end of a sprint normally implies that the work should not be counted toward that sprint's velocity.

Expected Benefits

- the Definition of Done provides a checklist which usefully guides pre-implementation activities: discussion, estimation, design
- the Definition of Done limits the cost of rework once a feature has been accepted as "done"
- having an explicit contract limits the risk of misunderstanding and conflict between the development team and the customer or product owner.

[Agile Alliance]

A general definition of done can be spelled out in a “done thinking grid”



[www.scrumalliance.org]

Backlogs

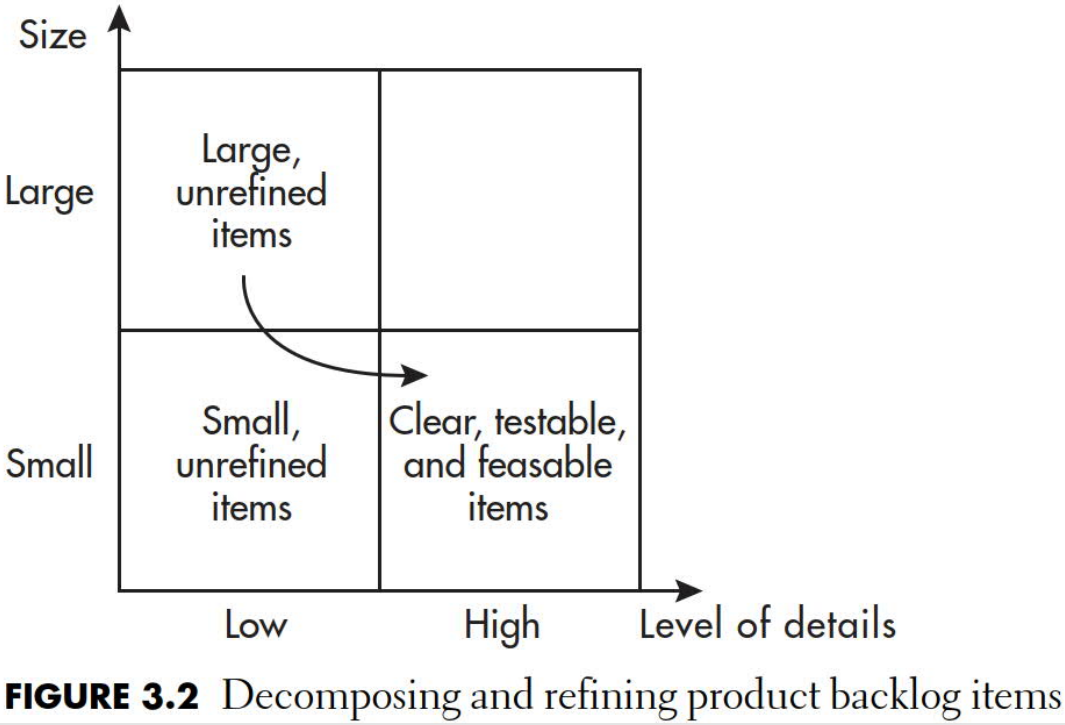
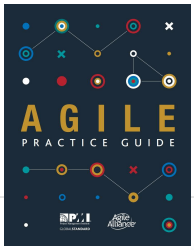


FIGURE 3.2 Decomposing and refining product backlog items

[Agile product management, 2010]



5.2.3 BACKLOG REFINEMENT

In iteration-based agile, the product owner often works with the team to prepare some stories for the upcoming iteration during one or more sessions in the middle of the iteration. The purpose of these meetings is to refine enough stories so the team understands what the stories are and how large the stories are in relation to each other.

There is no consensus on how long the refinement should be. There is a continuum of:

- Just-in-time refinement for flow-based agile. The team takes the next card off the to-do column and discusses it.
- Many iteration-based agile teams use a timeboxed 1-hour discussion midway through a 2-week iteration. (The team selects an iteration duration that provides them frequent-enough feedback.)
- Multiple refinement discussions for iteration-based agile teams. Teams can use this when they are new to the product, the product area, or the problem domain.

Technical debt

Technical Debt: good/bad

- Just as Financial debt, technical debt is not all bad
  - If it allows rapid delivery to elicit quick feedback and correct design
  - If the interest you have to pay is less than the benefit you gain by taking the debt
    - E.g. delaying testing of features that are rarely used
    - Sometimes you don't have to pay back the debt, it is pointless if the code won't be touched again.
- Bad
  - Messy code slows down the productivity and team morale
  - You have to make more ad-hoc fixes to keep up
  - If taken shortcuts are not fixed properly, i.e., debt is not repaid.

Different types of Debt (1/2)

- Design and Architecture debt → Using sub-optimal solutions, shortcuts and shortcomings
  - Fixing it - Future maintenance will be easy and productivity improves.
- Code debt → Unncecessary code duplication and complexity, every hack builds up bad code
  - Fixing it - Fewer defects, Happier developers and higher productivity.

Different types of Debt (2/2)

- Documentation debt → missing or not enough documentation of any type (e.g. maintenance guidelines, code comments)
  - Fixing it - Higher productivity.
- Defect debt → familiar defects that are not fixed
  - Fixing it - Happier developers and higher productivity.
- Testing debt → planned tests that were not run or drawbacks in test suite (e.g. low code coverage)
  - Fixing it - Better predictability and fewer surprises.

LUT
Lappeenranta
University of Technology

Jesse Yli-Huumo

THE ROLE OF TECHNICAL DEBT IN SOFTWARE DEVELOPMENT

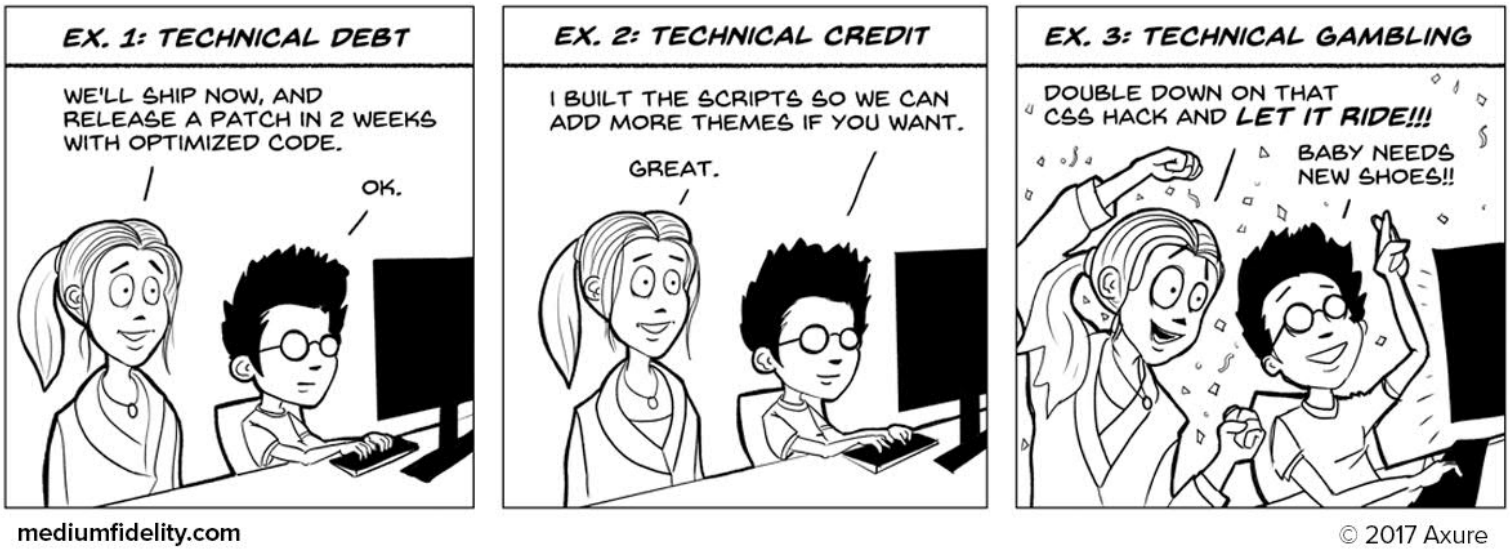
Thesis for the degree of Doctor of Science (Technology) to be presented with due permission for public examination and criticism in the Auditorium 2310 at Lappeenranta University of Technology, Lappeenranta, Finland on the 15th of June, 2017, at noon.

	Intentional reasons	Unintentional reasons
Technical reasons	*Code or design is too complex to work with. Intentional decision to take techical debt to avoid a long and complex solution. Could be a motivational issue or a goal to prevent breaking the software.*	*Developer uses a bad design or implements low quality code unknowingly due to lack of competence or coding guides available.*
Organizational reasons	*Product or feature time-to-market or company business needs force development team to take technical debt intentionally, because there is not enough time for proper implementation.*	*Software processes, company or team architectures, or other high-level organizational aspects involving people or tools are not designed or used properly, which causes technical debt in terms of decreased productivity or quality.*

Legacy reasons

Software ages after a time, which could be improved with new technologies, designs, and tools. However, older code and design can be seen as technical debt, when there is better solutions available.

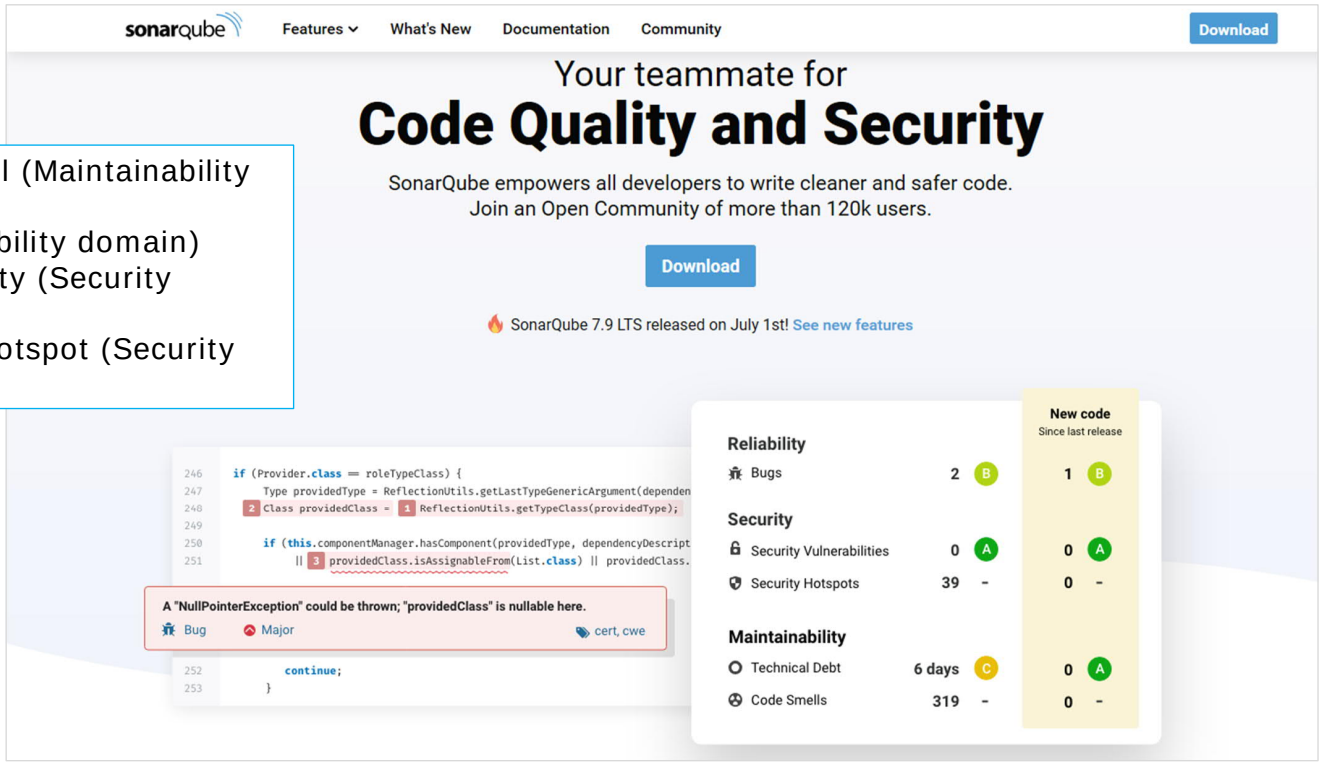
Figure 7. The reasons and causes for technical debt



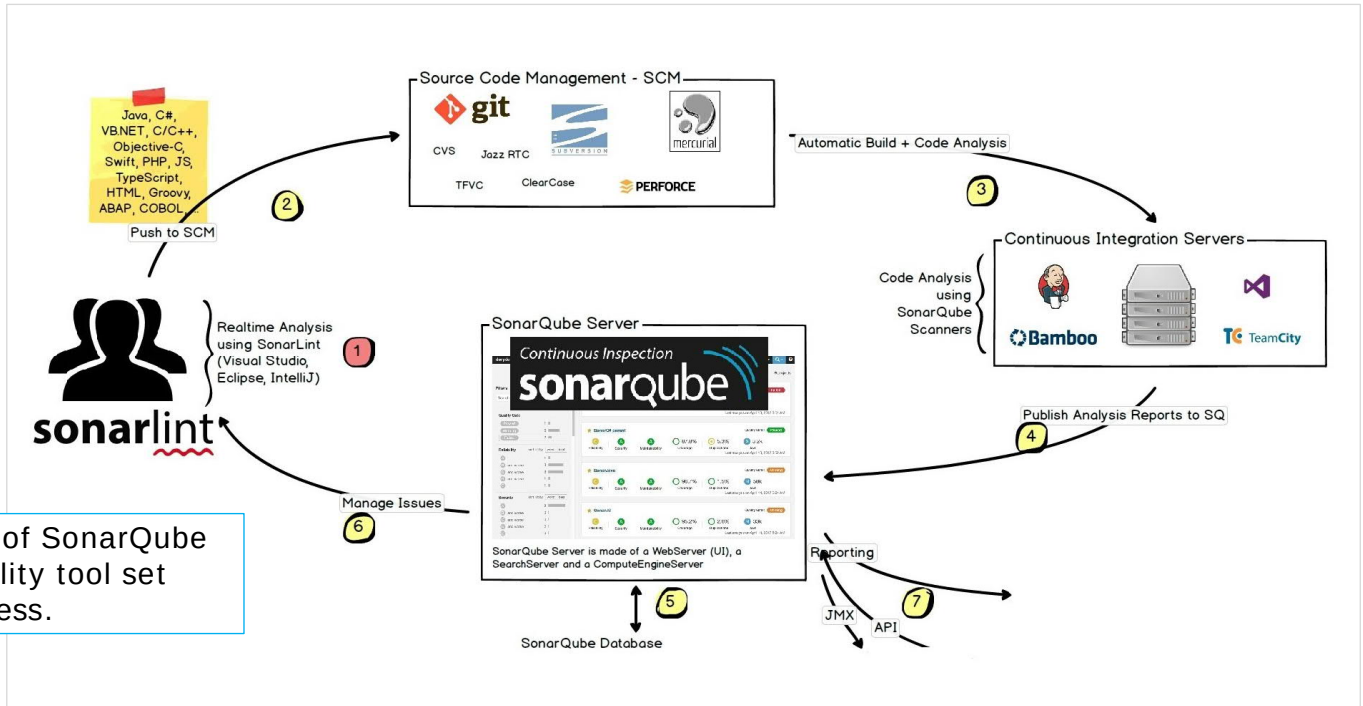
What is Technical Debt?

- Coined by Ward Cunningham in 1992 to communicate problems due to ”developing not in the right way”
- Analogous to financial debt
  - Financial debt = borrow money against a future date
  - Technical debt = borrow time against a future date
- Known issues (e.g. using proven pattens or thorough testing) delayed to meet a deadline or releasing to the market, that can cause problems in future if not completed.

- Code Smell (Maintainability domain)
- Bug (Reliability domain)
- Vulnerability (Security domain)
- Security Hotspot (Security domain)



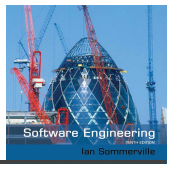
Static analysis tools may calculate technical debt



Example of SonarQube code quality tool set and process.

Extreme programming (XP)

Extreme programming practices (a)



Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (b)

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

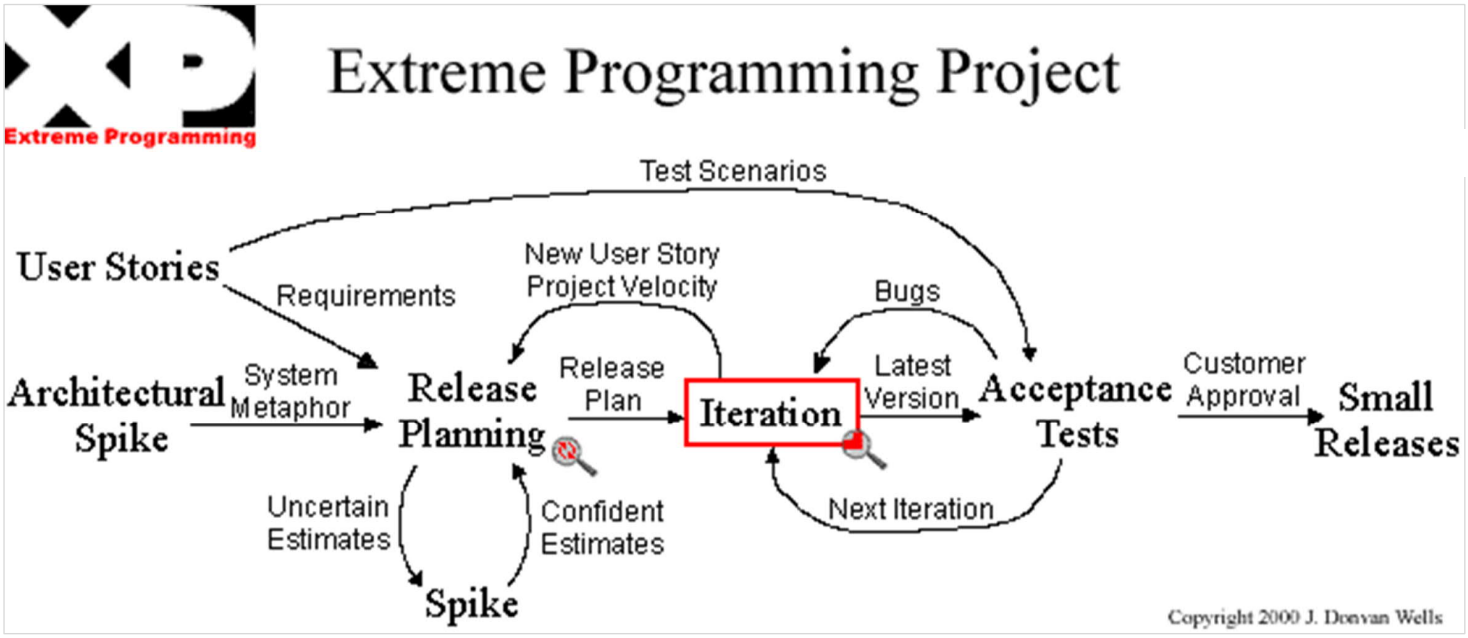
Influential XP practices

- Extreme programming has a technical focus and is not easy to integrate with management practice in most organizations.
- Consequently, while agile development uses practices from XP, the method as originally defined is not widely used.
- Key practices
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

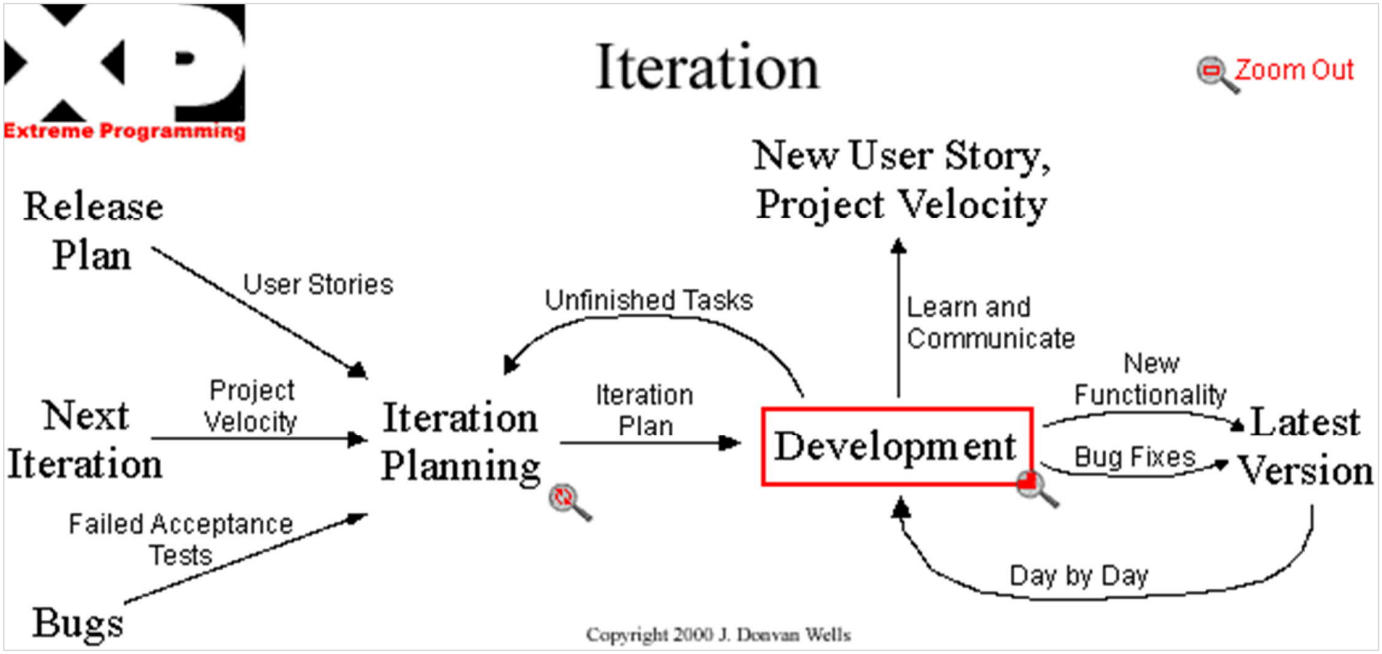
User stories for requirements

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as user stories or scenarios.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

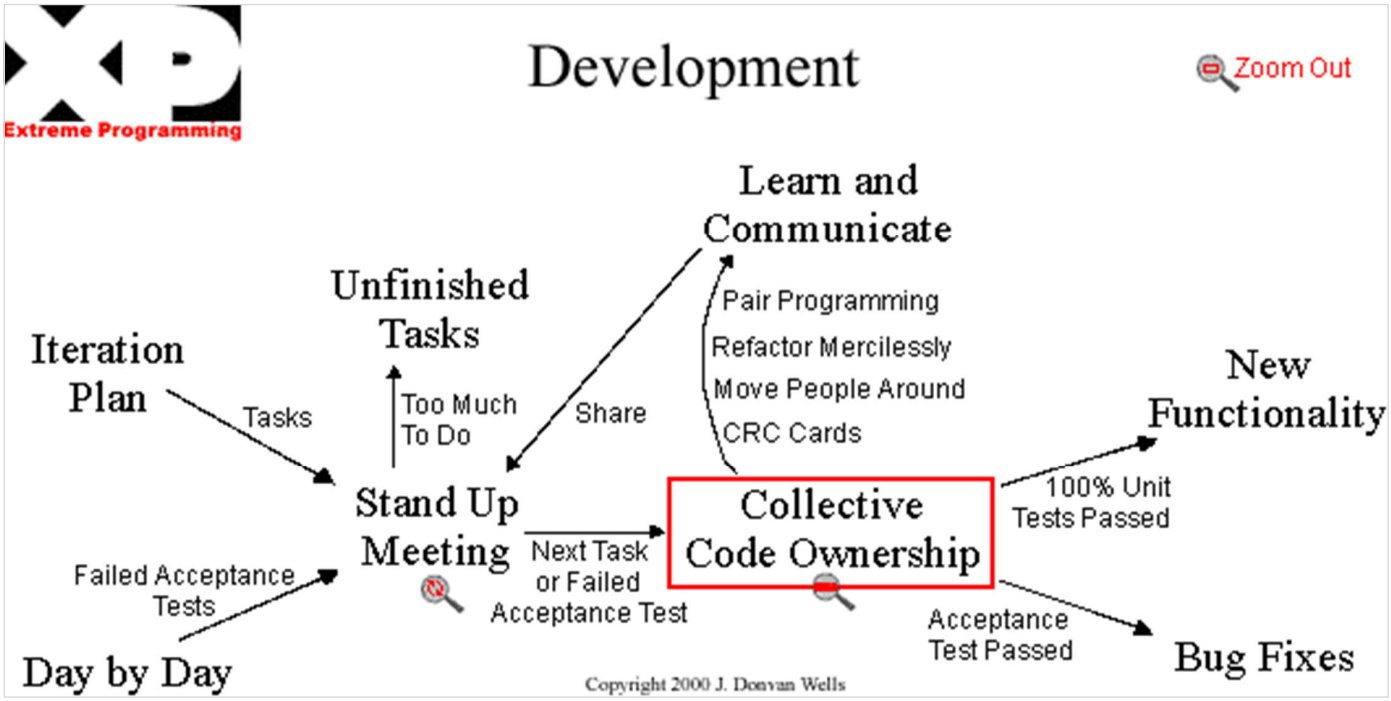
XP project



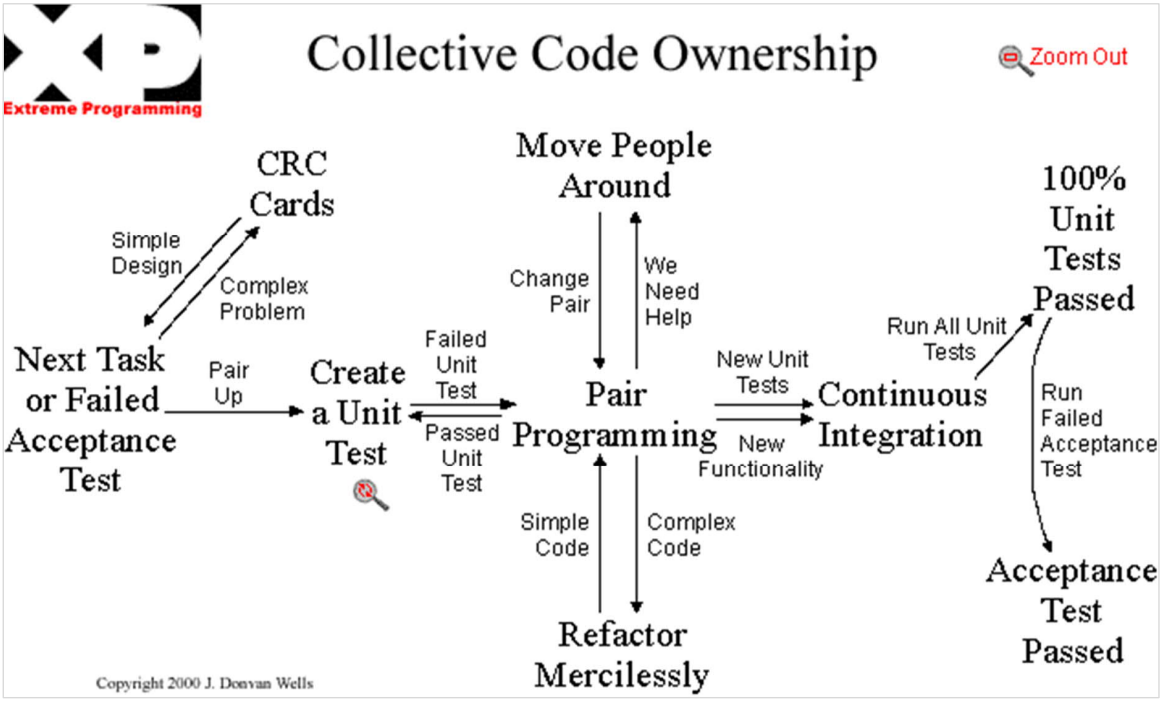
XP iteration



XP development



XP code



CRC = Class, Responsibilities, and Collaboration Cards

Tampereen yliopisto
Tampere University

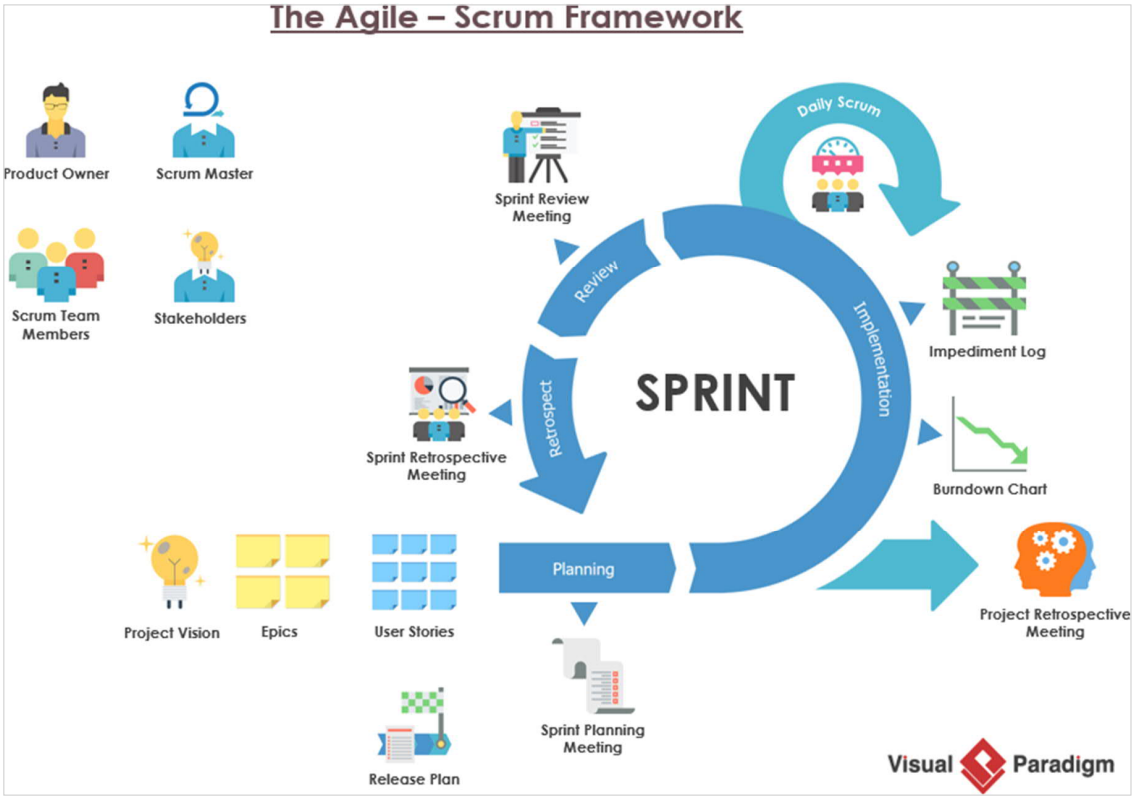
Scrum

Tampereen yliopisto
Tampere University

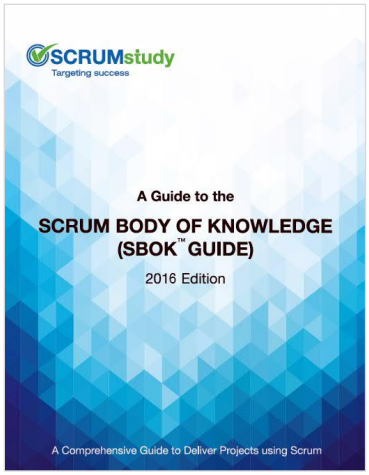
Scrum, basic terms

Product Backlog (PB) (FI: kehitysjono, kehityspino, tuotepino)
Sprint (FI: pyrähdys), iteration, some may call also as increment
Sprint Backlog (SB), work items to be done in the Sprint

Work item, e.g. features, user stories
Task (FI: tehtävä), work item is splitted to tasks (of size at most 1..1,5 work days)
Planning poker (helps in estimation = ”smart quesses”)
Velocity, ability to complete tasks (FI: vauhti, kehityskyvykkyys)
Scrum Master (FI: Scrummaster)
Product Owner (FI: tuoteomistaja)
Refactoring, fine-tuning (FI: refaktorointi, kodin hiominen, koodin parantelu)
Burndown Chart (or Burnup Chart) (FI: edistymiskäyrä, edistymisen seuranta)
Definition of Done (DoD), how you know when it is ready ? (FI: valmiin määritelmä).



[https://i.pinimg.com/originals/22/01/30/2201302dc021ce52427dea4c20479fc5.jpg]



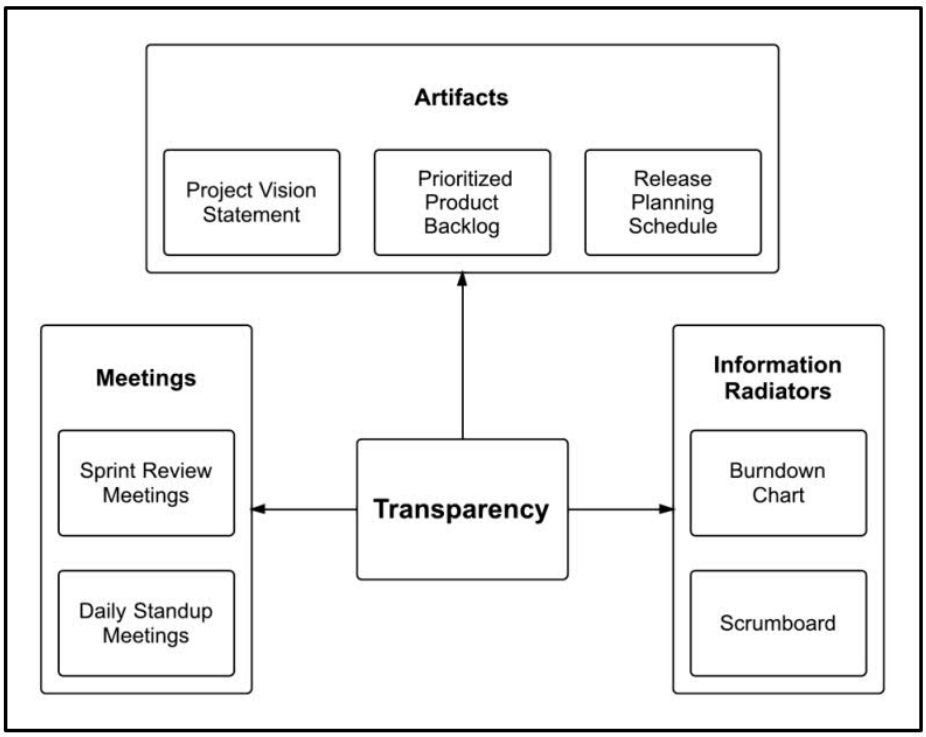
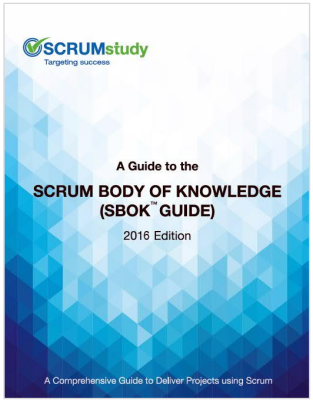


Figure 2-1: Transparency in Scrum



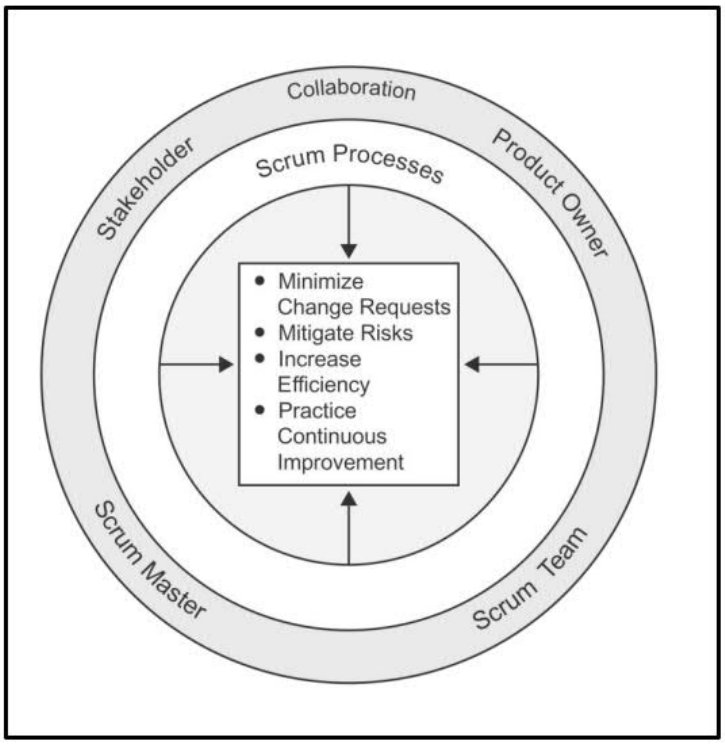
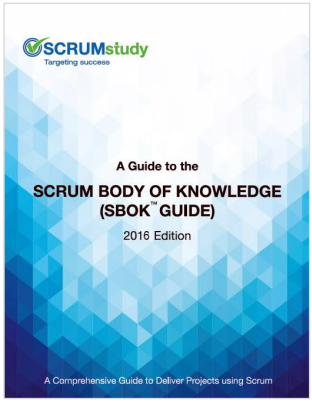


Figure 2-6: Benefits of Collaboration in Scrum Projects



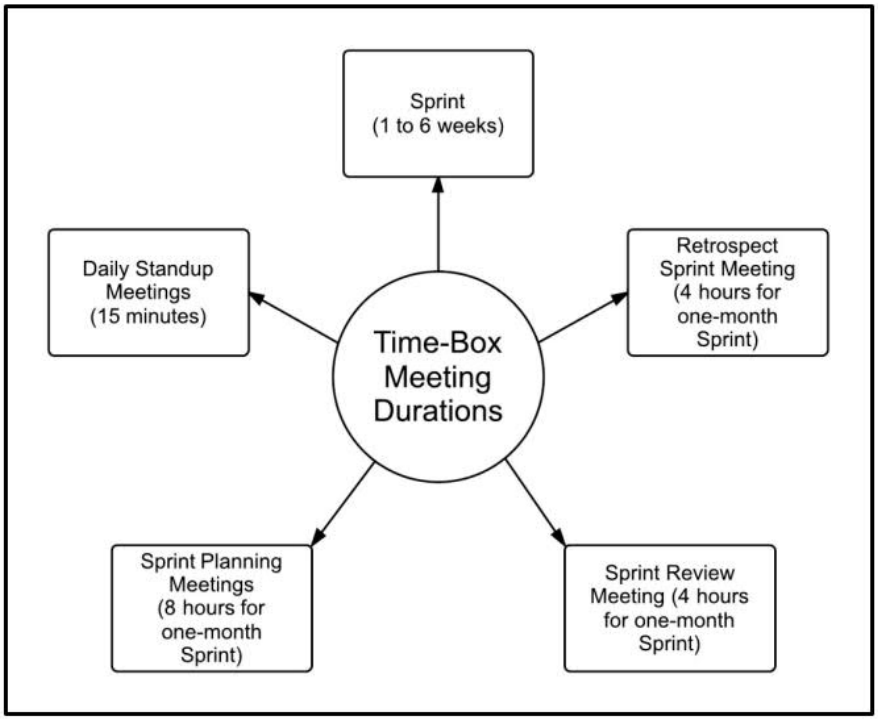
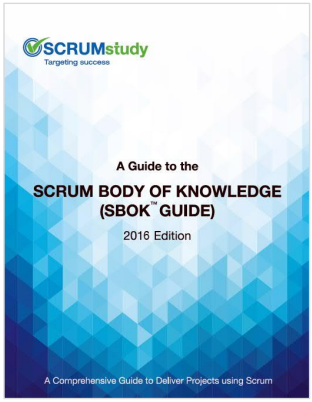


Figure 2-8: Time-Box Durations for Scrum Meetings



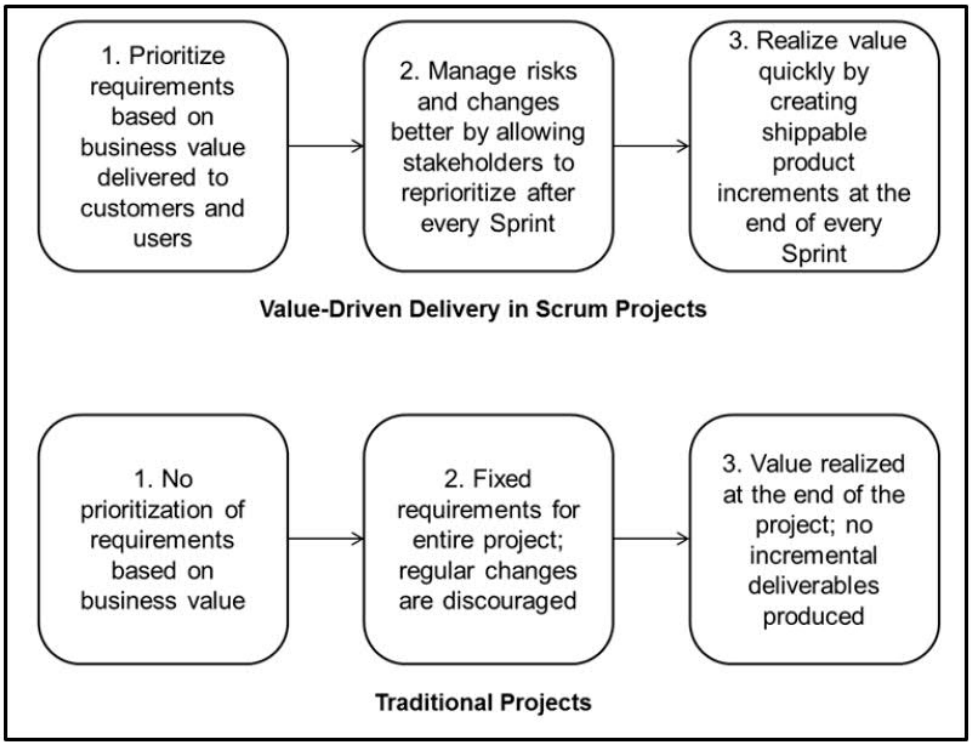
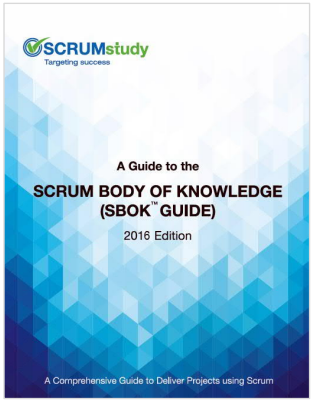


Figure 4-1: Delivering Value in Scrum vs. Traditional Projects



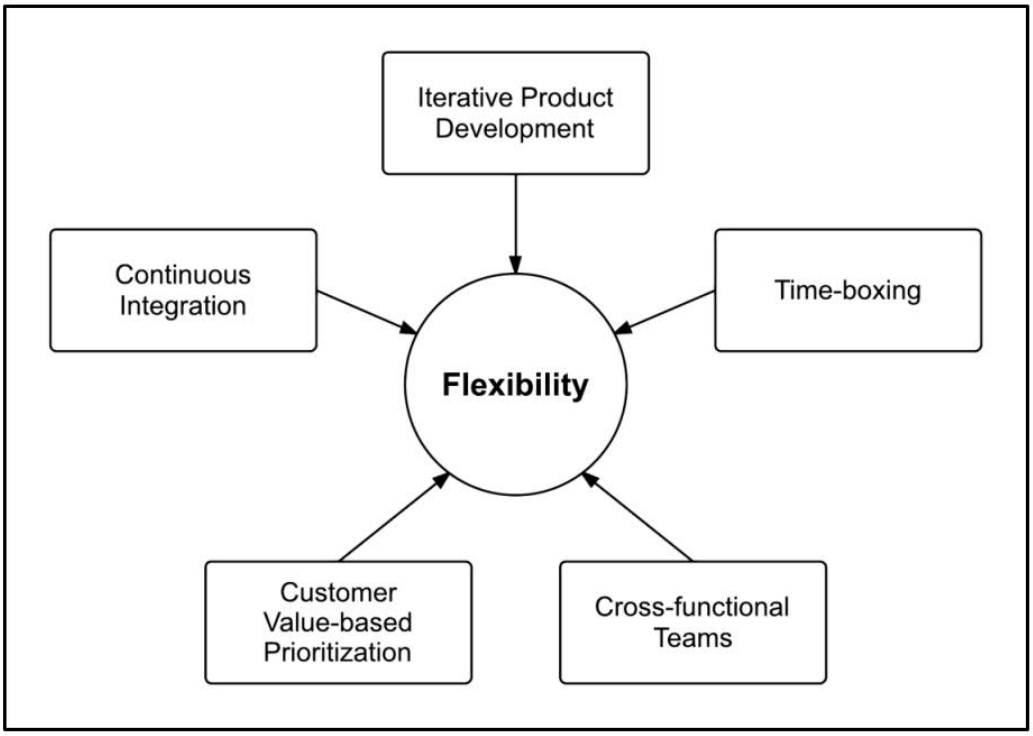


Figure 6-3: Scrum Characteristics for Achieving Flexibility

Scrum

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.
-

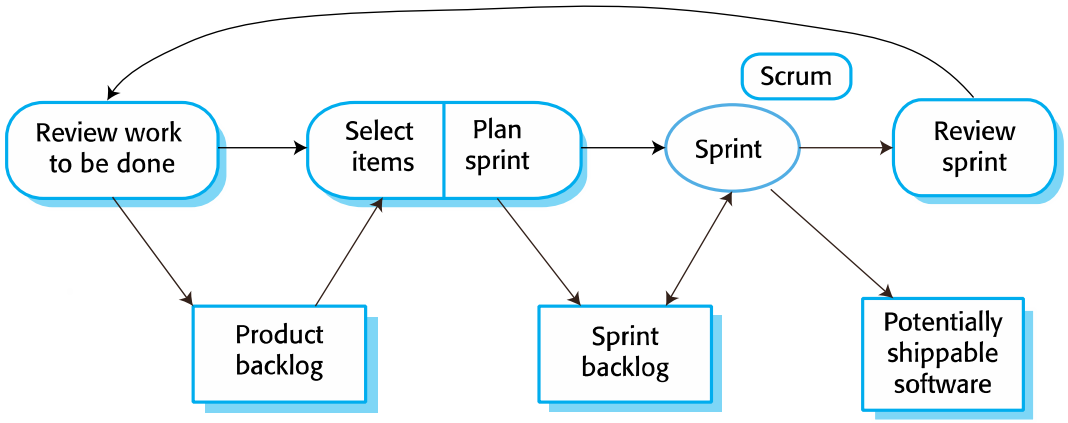
Scrum terminology (a)

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum terminology (b)

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum sprint cycle



The Scrum sprint cycle

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.

The Sprint cycle

- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Teamwork in Scrum

- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

Scrum Cheat Sheet

Product Owner
Owns the Product Backlog

The Product Owner represents the interests of everyone with a stake in the project (Stakeholder) and he is responsible for the final product.

- elicit product requirements
- manage the Product Backlog
- manage the release plan
- manage the Return on Investment

Sprint Planning
Commit the deliverable(s) to the PO

Two part meeting. First, the PO presents the User Stories. Second, when the Team thinks they have enough Stories to start the Sprint, they begin breaking it down in Tasks to fill the Sprint Backlog.

Timebox: 4 hours
Owner: Product Owner
Participants: Team, Scrum Master

Product Backlog
Dynamic prioritized list of requirements

The requirements for the product are listed in the Product Backlog. It is an always changing, dynamically prioritized list of requirements ordered by Business Value. Requirements are broken down into User Stories by the PO.

Prioritize the requirements by playing the Business Value game.

Buy these at www.agile42.com

Scrum Master
Owns the Scrum process

The Scrum Master is responsible for the Scrum process.He ensures everybody plays by the rules. He also removes impediments for the Team. The Scrum Master is not part of the Team.

- manage the Scrum process
- remove impediments
- facilitate communication

Daily Scrum
Inspect and Adapt the progress

In this standup meeting the Team daily inspects their progress in relation to the Planning by using the Burndown Chart, and makes adjustments as necessary.

Timebox: 15 minutes
Owner: Scrum Master
Participants: Team, all interested parties may silently attend.

Burndown Chart
Estimated remaining time of the Sprint

The Burndown chart shows the amount of work remaining per Sprint. It is a very useful way of visualizing the correlation between work remaining at any point in time and the progress of the Team(s).

Use a tool such as Agilo to automatically create the Burndown Chart.

Learn more at www.agile42.com

Team Member
Owns the software

The team figures out how to turn the Product Backlog into an increment of functionality within a Sprint. Each team member is jointly responsible for the success of each iteration and of the project as a whole.

- software quality
- technical implimentation of User Stories
- delivery of functional software increment
- to organize themselves

Retrospective
Maintain the good, get rid of the bad

At the end of a Sprint, the Team evaluates the finished Sprint. They capture positive ways as a best practice, identify challenges and develop strategies for improvements.

Timebox: 2 hours
Owner: Scrum Master
Participants: Team, Product Owner, optionally the PO can invite Stakeholders

Sprint Backlog
List of committed User Stories

The Sprint Backlog contains all the committed User Stories for the current Sprint broken down into Tasks by the Team. All items on the Sprint Backlog should be developed, tested, documented and integrated to fulfil the commitment.

Estimate Story complexity by playing Planning Poker.

Buy these at www.agile42.com

Requirements
Make SMART Requirements: Simple, Measurable, Achievable, Realistic, Traceable.

User Stories
INVEST in User Stores: Independant, Negotiable, Valuable, Estimatable, Small, Traceable.

Tasks
Make sure a Task is TECH: Time boxed, Everybody (can pick it up), Complete and Human-readable.

SCRUM RULES REFERENCE SHEET

Required Rules to Start – the "Agile Skeleton":

- ScrumMaster Identified and Team Members Available to DoWork
- Team Agrees to Demonstrate Working Software in No More Than 30 Days
- Stakeholders Invited to Demonstration

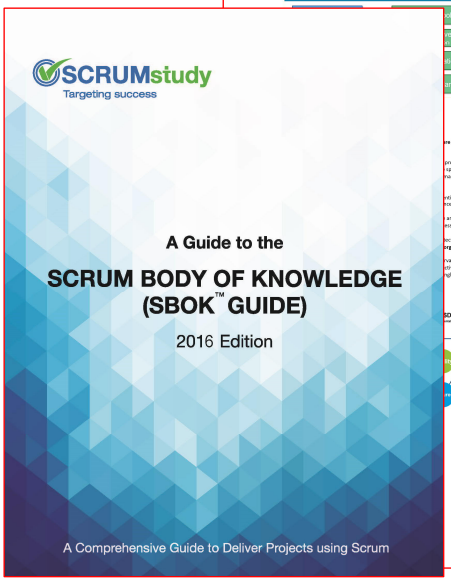
Basic Rules of Scrum to Implement As Soon As Possible:

- Full-Time ScrumMaster w/ Authority to Implement Rules of Scrum
- Full-Time Product Owner (with Expertise and Authority) Identified
- Cross-Functional Team Including ScrumMaster and Product Owner
- Team Size 7 +/-2, Maximum of 12
- Product Owner Works With Team and All Other Stakeholders
- Product Backlog Created and Managed by Product Owner
- Daily Scrum Meeting (Questions: Completed? Will Complete? Obstacles?)
- Daily Scrum at Same Place and Time and Less Than 15 Minutes
- All Team Members Required at Daily Scrum
- Anyone Can Observe Daily Scrum, but Not Participate
- Sprint Length No More Than 30 Days, and Consistently Same Length
- Sprint Planning Meeting with Whole Team
- 1st Part of Sprint Planning: Product Backlog Items Selected by Team
- 2nd Part of Sprint Planning: Team Creates Sprint Backlog of Estimated Tasks
- Sprint Backlog Tasks Added/Updated/Removed by Team
- Sprint Burndown Chart
- Retrospective Meeting with Whole Team for Process Improvements
- Definition of "Done"
- Commitment Velocity Calculated (from Sprint Backlog Estimates)
- Team Members Volunteer for Tasks, 1 Task at a Time Until Complete
- Team can Seek Advice, Help, Info
- ScrumMaster Tracking and Removing Obstacles
- No Interruptions or Reprioritization of Team's Work During Sprints
- No "Break" Between Sprints
- Sustainable Pace – Timebox Effort, Not Just Schedule
- Quality is Not Negotiable – Defects Go on Top of Product Backlog
- Sprint Planning and Review Meetings 1/20th Sprint Duration

TRUTHFULNESS IS THE FOUNDATION!

Berteig Consulting

Scrum cheatsheet



Practical problems with agile methods

- The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
- Agile methods are most appropriate for new software development rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.
- Agile methods are designed for small co-located teams yet much software development now involves worldwide distributed teams.

Contractual issues

- Most software contracts for custom systems are based around a specification, which sets out what has to be implemented by the system developer for the system customer.
- However, this precludes interleaving specification and development as is the norm in agile development.
- A contract that pays for developer time rather than functionality is required.
  - However, this is seen as a high risk my many legal departments because what has to be delivered cannot be guaranteed.

Agile methods and software maintenance

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

Agile maintenance

- Key problems are:
  - Lack of product documentation
  - Keeping customers involved in the development process
  - Maintaining the continuity of the development team
- Agile development relies on the development team knowing and understanding what has to be done.
- For long-lifetime systems, this is a real problem as the original developers will not always work on the system.

Agile and plan-driven methods

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

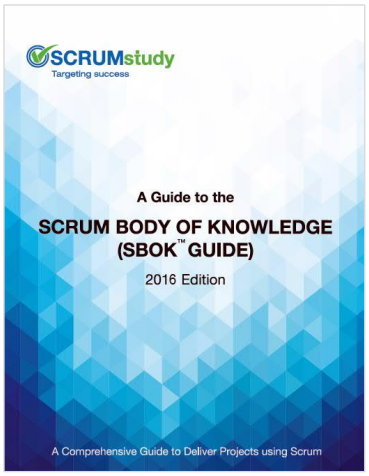
Tampereen yliopisto
Tampere University

Epic

Epic(s) are written in the initial stages of the project when most User Stories are high-level functionalities or product descriptions and requirements are broadly defined.

They are large, unrefined User Stories in the Prioritized Product Backlog.

[SBOK Guide 2016]



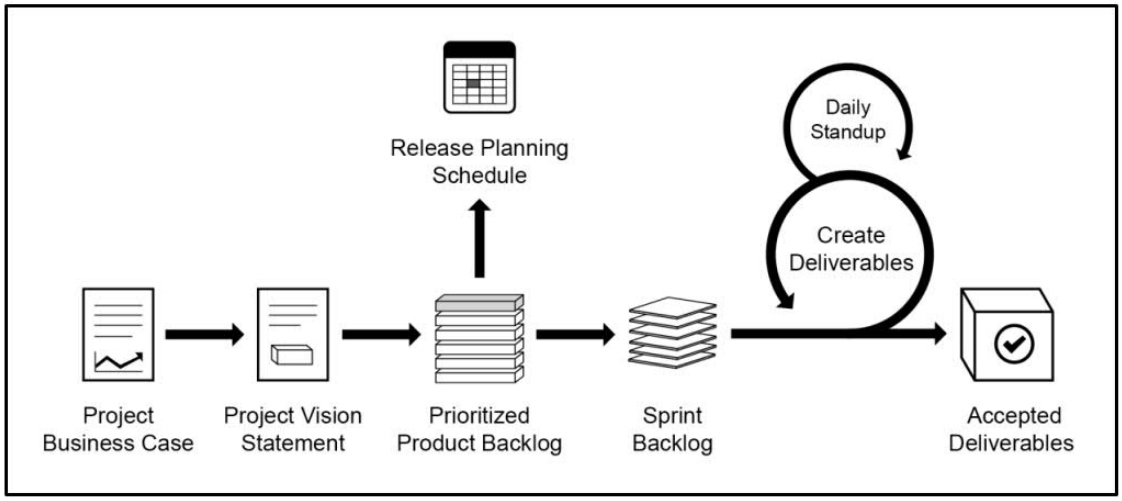
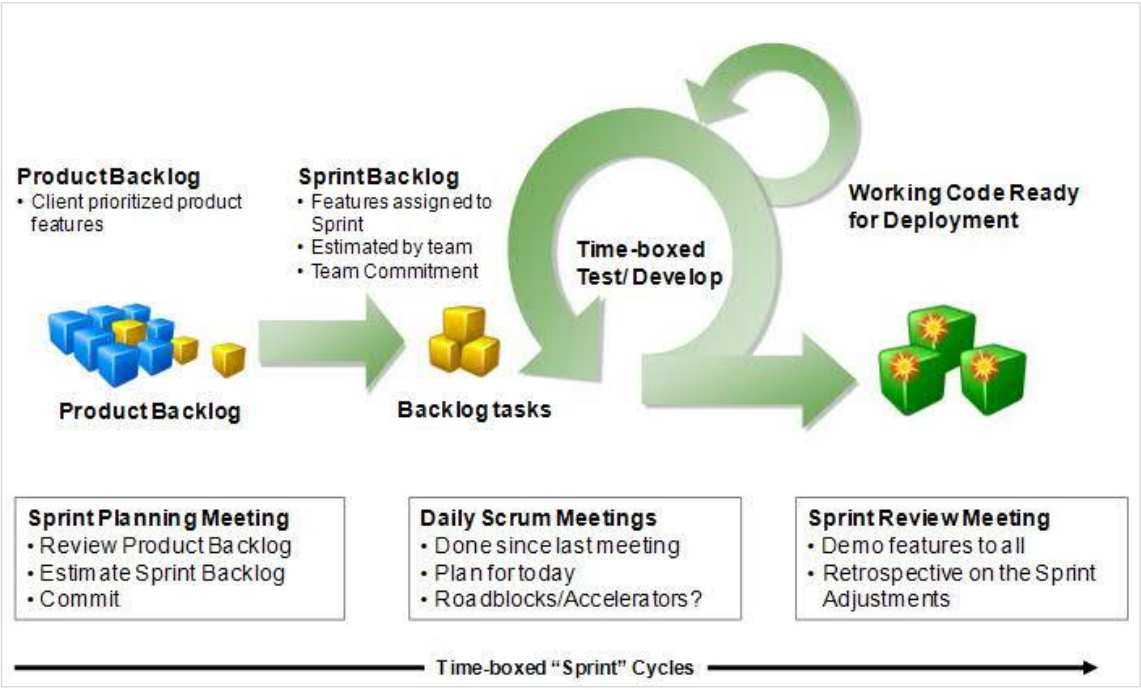


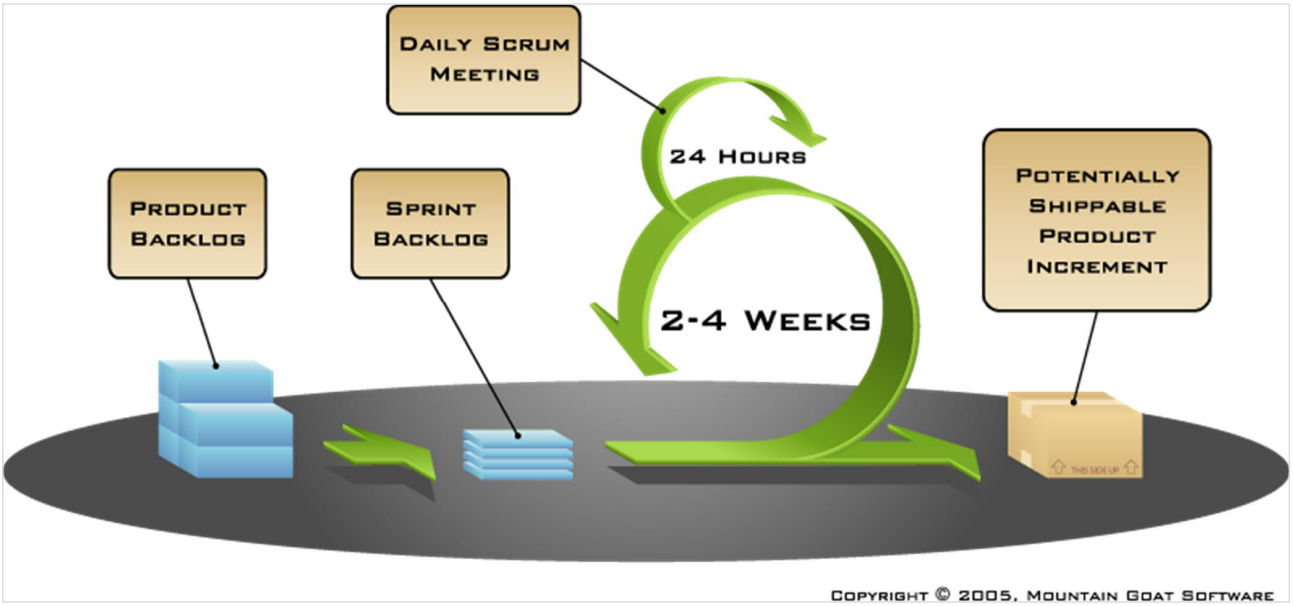
Figure 1-1: Scrum Flow for one Sprint

Scrum basics



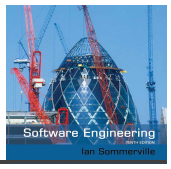
Sprint retrospective meeting

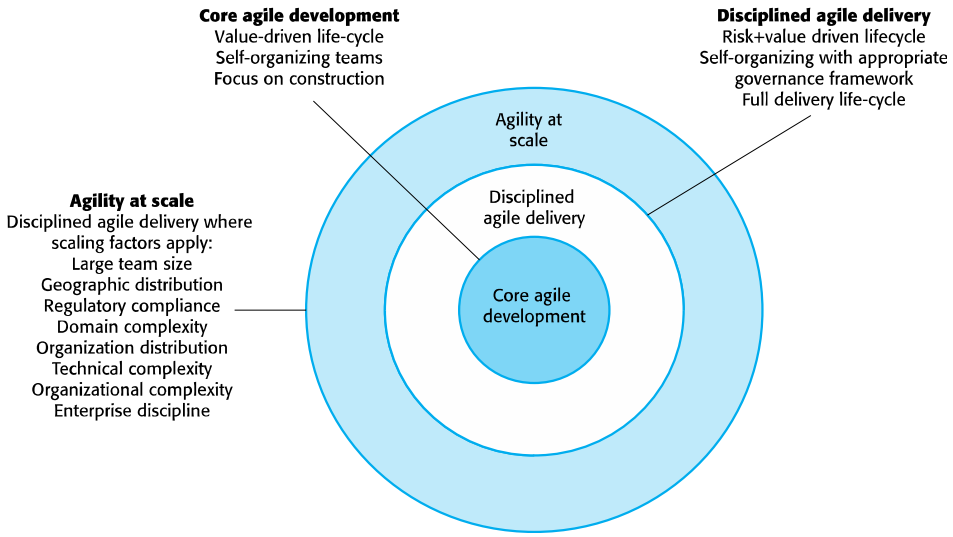
Putting it all together



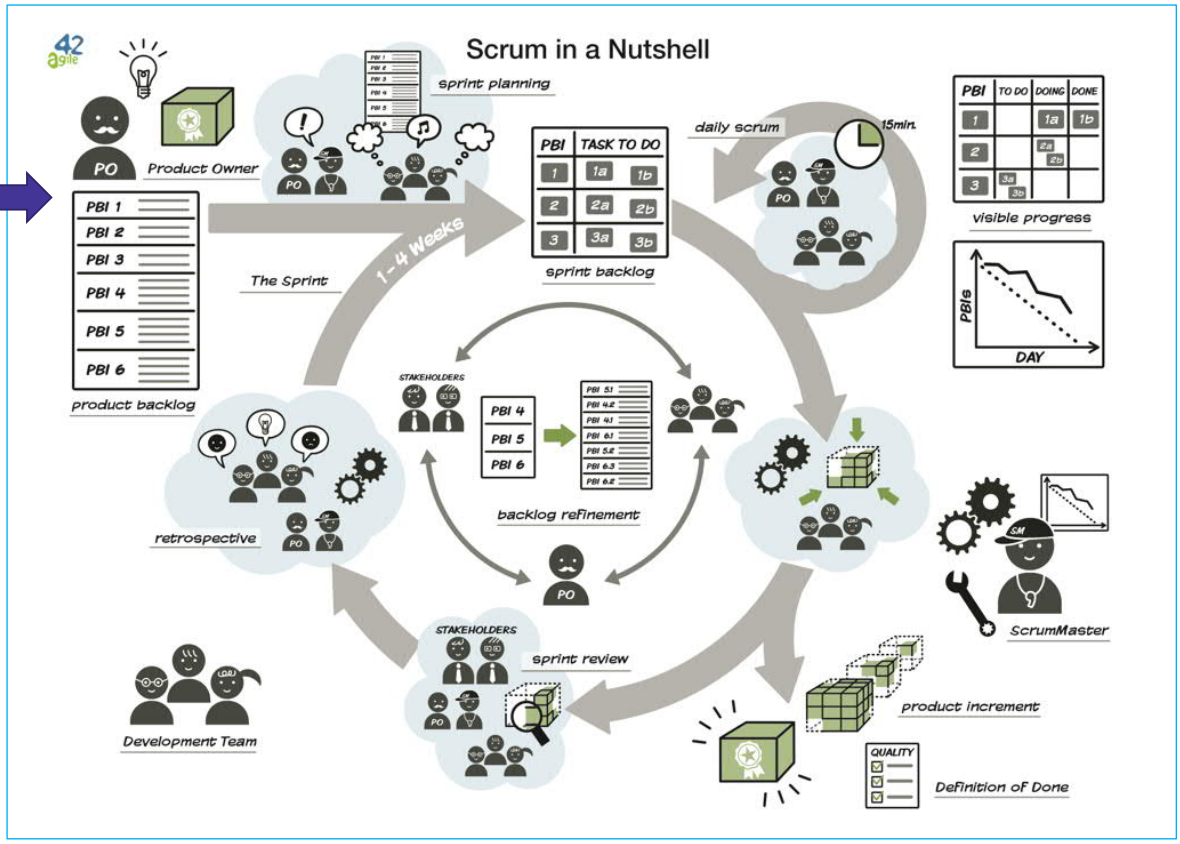
[www.mountaingoatsoftware.com/scrum]

IBM's agility at scale model





Scrum

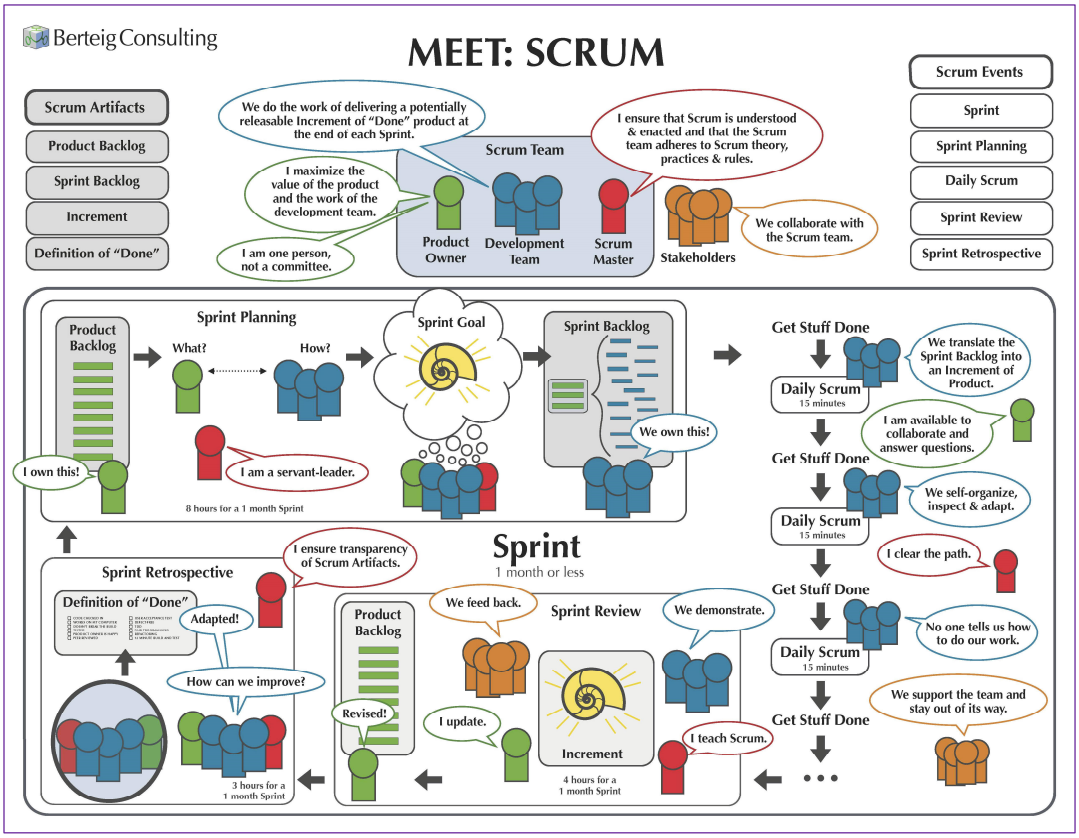


Sprint planning

Sprint work

Sprint review

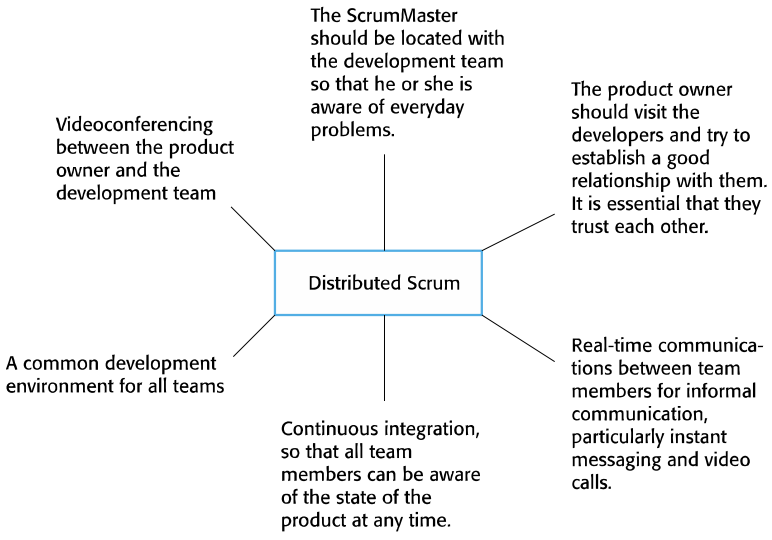
Sprint retrospective



Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Distributed Scrum



Tampereen yliopisto
Tampere University

Scrumboard

Tampereen yliopisto
Tampere University

Wrong use of agile or Scrum

Tampereen yliopisto
Tampere University

5 Ways To Fail With Scrum

(https://www.thescrummaster.co.uk/scrum/5-ways-to-fail-with-scrum/)

- Not Producing A “Done” Increment By The End Of A Sprint
- Lack of Product Backlog Refinement (remember time box)
- Scrum Master As The Development Team Manager (delf-guiding team)
- No Cross Functional Teams (wide skills set is better)
- Lack Of Agility From Fixing Scope, Budget And Timeline (agile…).

How Not To Fail

These types of scenarios will occur on many projects. The secret to solving them is to use the inspection, adaptation and transparency that comes with Scrum to highlight and fix these dysfunctions as they are identified. If the organisation is serious and committed to becoming Agile, then this is the only way to be more successful.

Tampereen yliopisto
Tampere University

How to Make Scrum Fail

Glenn Dejaeger, Jun. 20, 2019

- No or bad retrospectives
- Bad Product Owner
- Bad Scrum Master
- Scrum Ceremonies are taking too long (e.g. daily stand-up)
- Wrong definition of done
- No velocity tracking
- Waterfall within sprint
- Technical debt
- Interruptions / PO bypassed
- No analysis / documentation.

Scrum is not a silver bullet. It’s no methodology that defines a lot of rules. Instead it’s a framework which defines some best practices. Of course you’ll have to adopt the basic principles to make it work.

Tampereen yliopisto
Tampere University

10 Reasons Why You Are Doomed to Fail With Scrum

Willem-Jan Ageling, Jun 14, 2020

1. No focus on value (deliver value to customers)
2. Unwillingness to change
3. Scrum Team doesn't interact with key stakeholders
4. Product Owner has insufficient power (PO: what to build)
5. Failing to address different ideas about Scrum (there are many ways to misunderstand Scrum's roles, artefacts, events and rules)
6. Scrum Master doesn't have Scrum knowledge
7. Development Team can't determine how to build the Increment (too strict rules)
8. Development Team can't build an increment in a Sprint (inspect, adapt)
9. No commitment
10. Your product environment isn't suited for Scrum.

Tampereen yliopisto
Tampere University

8 Reasons Why Agile Projects Fail

by Lee Cunningham
[https://www.agilealliance.org/8-reasons-why-agile-projects-fail/]

#1 Lack of Experience with Agile Methods
#2 Company Philosophy or Culture at Odds with Core Agile Values
#3 Lack of Management Support
#4 External Pressure to Follow Traditional Waterfall Processes
#5 Lack of Support for Cultural Transition
#6 A Broader Organizational or Communications Problem
#7 Unwillingness of Team to Follow Agile
#8 Insufficient training.

7.5% of the respondents told us that none of their agile projects could be considered "unsuccessful." Which is great news. (2015)

Tampereen yliopisto
Tampere University

12 brilliant ways to fail with Agile

(Willem-Jan Ageling, May 22, 2018)

1. Don’t focus on value, focus on velocity instead
2. Don’t allow any changes to your Sprint backlog
3. Don’t break down your items into smaller pieces, so you are sure to deliver later
4. Avoid Business and Software Development to work together
5. Micro-manage your development team’s performance
6. Avoid face-to-face communication
7. Build a Feature Factory (feature creep to optional items)
8. Make working overtime to make the Sprint a standard practice
9. Deliver stuff, worry about the quality later
10. Create complex solutions (architecture specifications)
11. Discourage the teams to self-organize
12. Skip inspect and adapt.

Tampereen yliopisto
Tampere University

7 simple ways to fail at agile

[https://www.cio.com/article/3234366/7-simple-ways-to-fail-at-agile.html]

1. Plan loosely and chaotically
2. Form an unstable, poorly selected team
3. Communicate as cryptically and infrequently as possible
4. Don't fully understand the project's scope or focus
5. Test poorly and haphazardly (ad-hoc / random testing)
6. Fail to win management and staff support
7. Disregard customer feedback.

"Success has many fathers, while failure is an orphan" is a saying frequently cited by contemplative business leaders. Yet when it comes to failed agile initiatives, the old bromide might be updated to state: "Success is a team effort, while failure is simply everybody's fault.“

Tampereen yliopisto
Tampere University

How To Fail With Agile (1/3): Twenty Tips to Help You Avoid Success

[https://www.mountaingoatsoftware.com/articles/how-to-fail-with-agile]

GUIDELINE 1: ***Don't trust the team or agile. Micromanage both your team members and the process.***

GUIDELINE 2: ***If agile isn't a silver bullet, blame agile.***

GUIDELINE 3: ***Equate self-managing with self-leading and provide no direction to the team whatsoever.***

GUIDELINE 4: ***Ignore the agile practices*****.**

GUIDELINE 5: ***Undermine the team's belief in agile.***

GUIDELINE 6: ***Continually fail to deliver what you committed to deliver during iteration planning.***

GUIDELINE 7: ***Cavalierly move work forward from one iteration to the next. It's good to keep the product owner guessing about what will be delivered.***

Tampereen yliopisto
Tampere University

How To Fail With Agile (2/3): Twenty Tips to Help You Avoid Success

[https://www.mountaingoatsoftware.com/articles/how-to-fail-with-agile]

GUIDELINE 8: ***Do not create cross-functional teams. Put all the testers on one team, all the programmers on another, and so on.***

GUIDELINE 9: ***Large projects need large teams. Ignore studies that show productivity decreases with large teams due to increased communication overhead. Since everyone needs to know everything, invite all fifty people to the daily standup.***

GUIDELINE 10: ***Don't communicate a vision for the product to the agile team or to the other stakeholders.***

GUIDELINE 11: ***Don't pay attention to the progress of each iteration and objectively evaluate the value of that progress.***

GUIDELINE 12: ***Replace a plan document with a plan "in your head" that only you know.***

GUIDELINE 13: ***Have one person share the roles of ScrumMaster (agile coach) and product owner. In fact, have this person also be an individual contributor on the team.***

Tampereen yliopisto
Tampere University

How To Fail With Agile (3/3): Twenty Tips to Help You Avoid Success

[https://www.mountaingoatsoftware.com/articles/how-to-fail-with-agile]

GUIDELINE 14: ***Start customizing an agile process before you've done it by the book.***

GUIDELINE 15: ***Drop and customize important agile practices before fully understanding them.***

GUIDELINE 16*: Slavishly follow agile practices without understanding their underlying principles.*

GUIDELINE 17: ***Don't continually improve.***

GUIDELINE 18: ***Don't change the technical practices.***

GUIDELINE 19: ***Rather than align pay, incentives, job titles, promotions, and recognition with agile, create incentives for individuals to undermine teamwork and shared responsibility.***

GUIDELINE 20: ***Convince yourself that you'll be able to do all requested work, so the order of your work doesn't matter.***

Tampereen yliopisto
Tampere University

Scrum roles

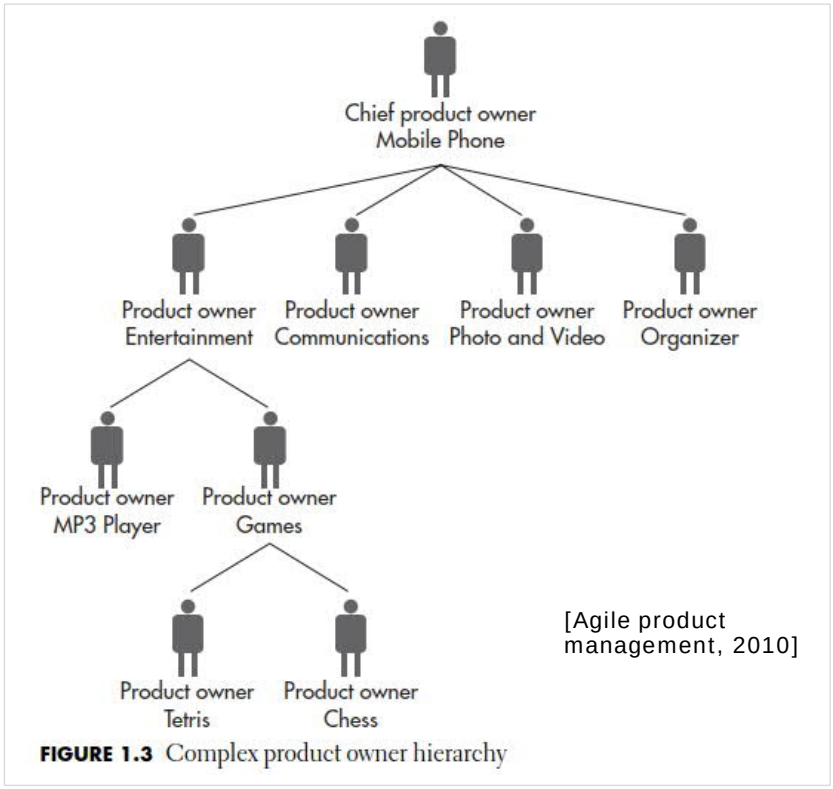
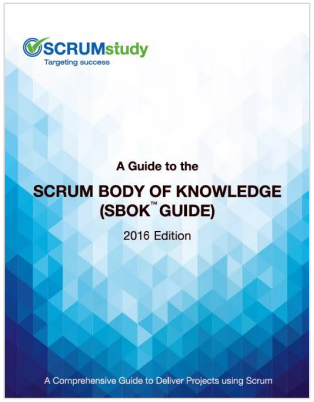


FIGURE 1.3 Complex product owner hierarchy

[Agile product management, 2010]

Scrum master is...

some kind of...
- facilitator
- problem solver
- equaliser
- and more.



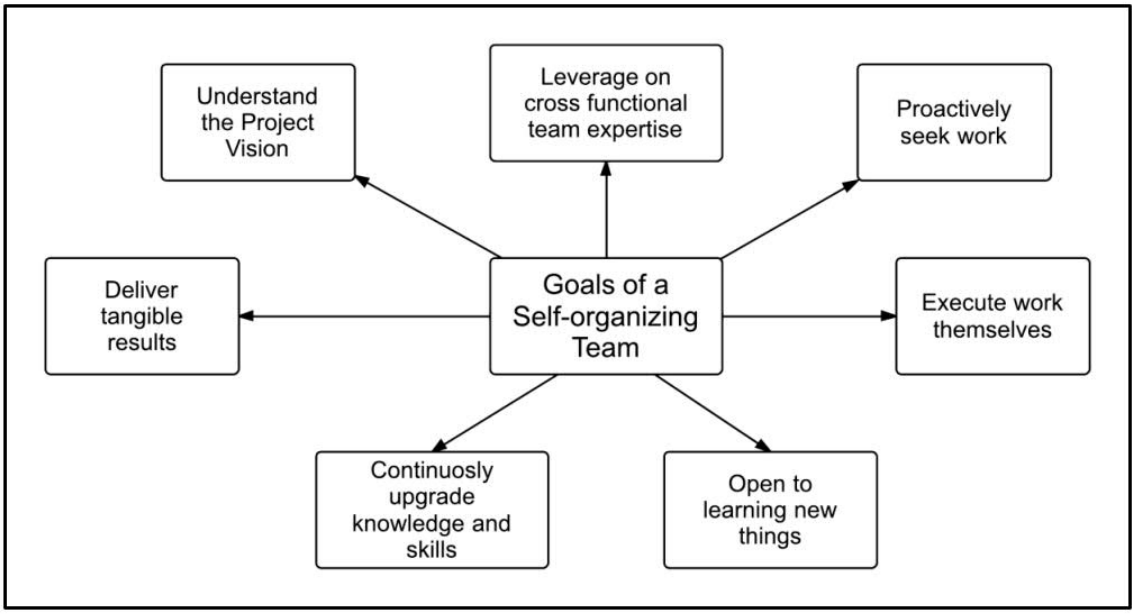
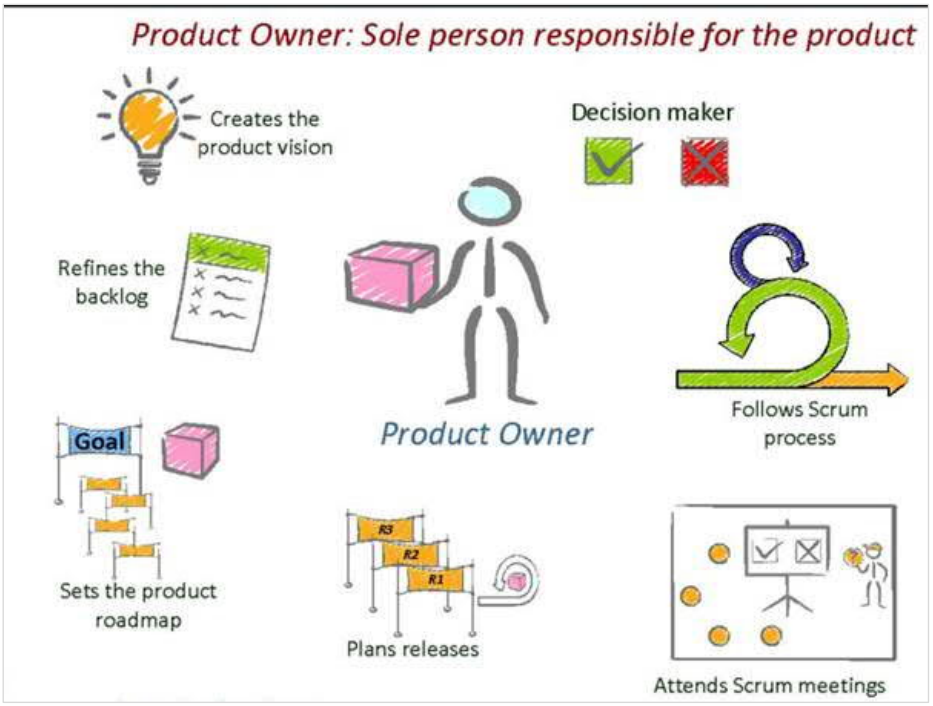
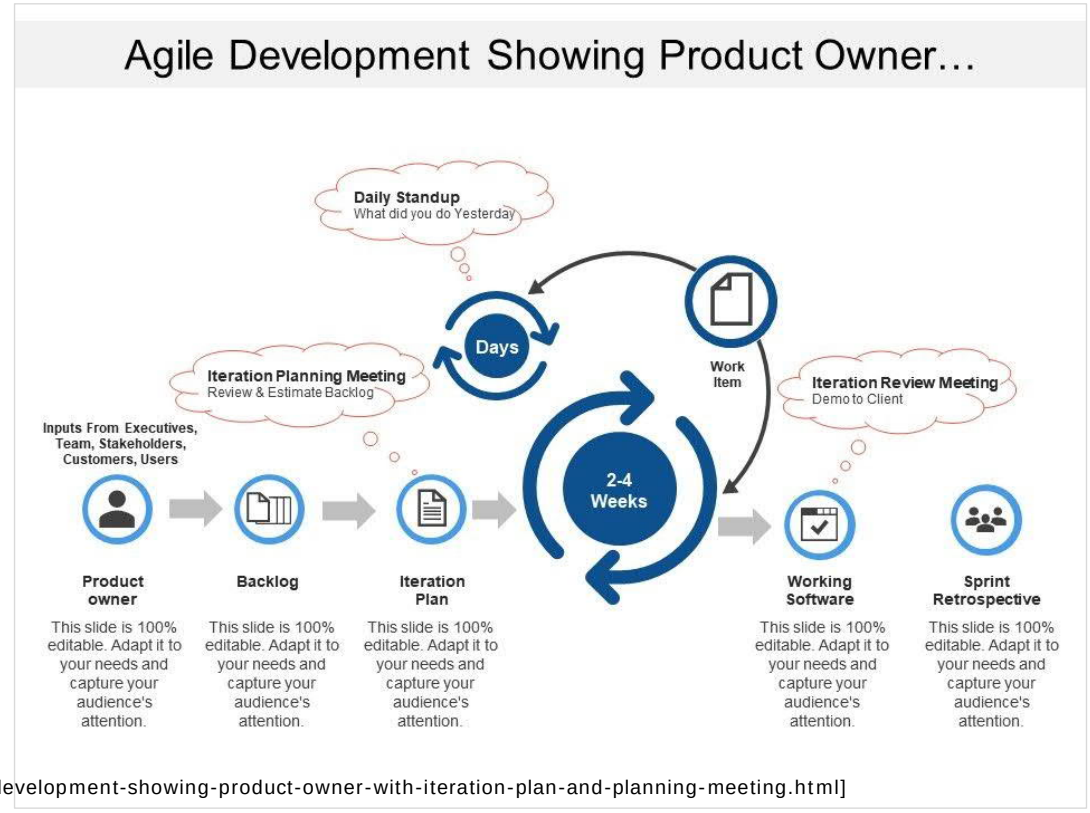


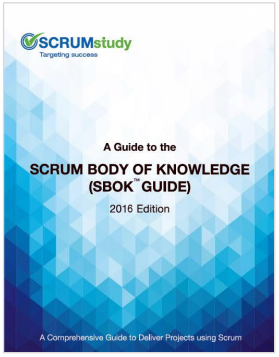
Figure 2-5: Goals of a Self-Organizing Team



[https://www.c-sharpcorner.com/article/product-owner-role-and-responsibilities/]



[www.slideteam.net/agile-development-showing-product-owner-with-iteration-plan-and-planning-meeting.html]



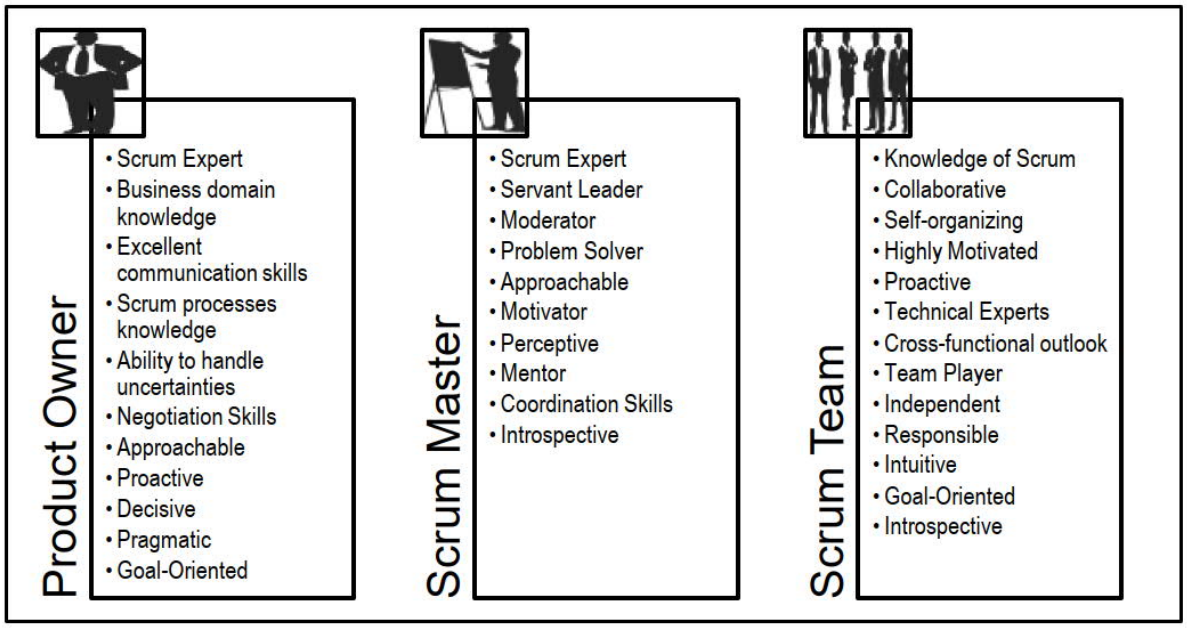


Figure 3-3: Desirable Traits for the Core Scrum Roles

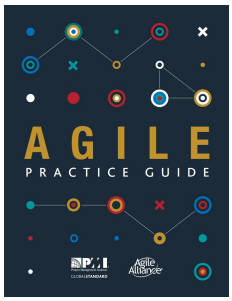
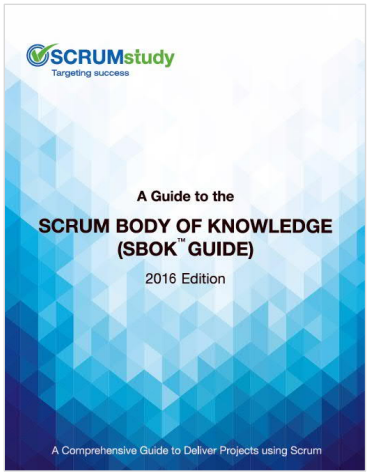


Table 4-1. Attributes of Successful Agile Teams

Attribute	Goal
Dedicated people	• Increased focus and productivity • Small team, fewer than ten people
Cross-functional team members	• Develop and deliver often • Deliver finished value as an independent team • Integrate all the work activities to deliver finished work • Provide feedback from inside the team and from others, such as the product owner
Colocation or ability to manage any location challenges	• Better communication • Improved team dynamics • Knowledge sharing • Reduced cost of learning • Able to commit to working with each other
Mixed team of generalists and specialists	• Specialists provide dedicated expertise and generalists provide flexibility of who does what • Team brings their specialist capabilities and often become generalizing specialists, with a focus specialty plus breadth of experience across multiple skills
Stable work environment	• Depend on each other to deliver • Agreed-upon approach to the work • Simplified team cost calculations (run rate) • Preservation and expansion of intellectual capital



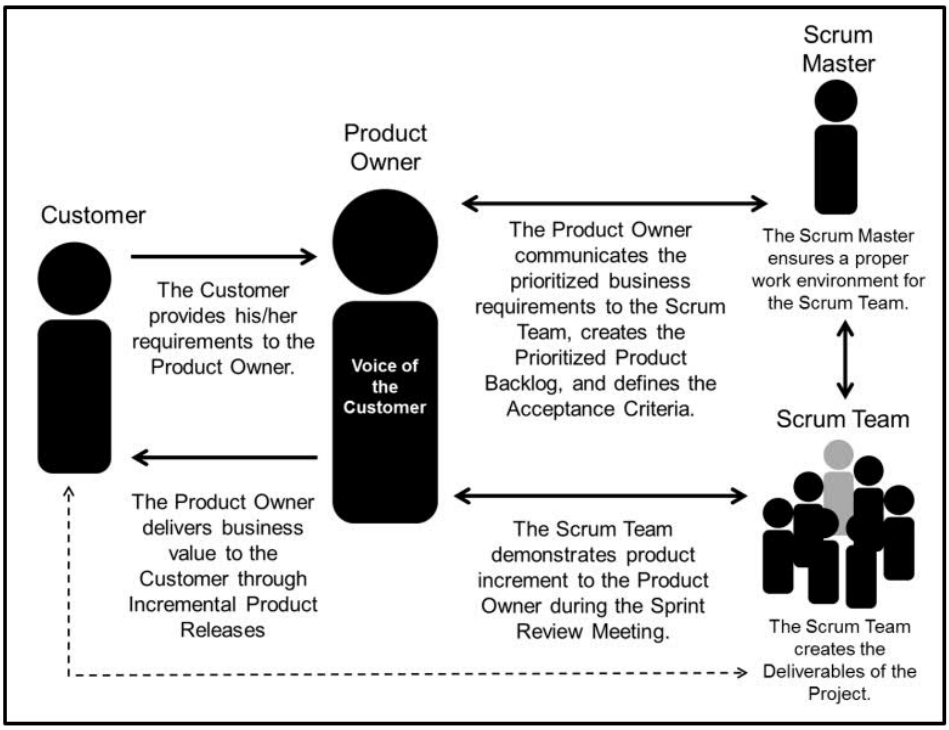


Figure 1-4: Organization in Scrum

Tampereen yliopisto
Tampere University

Differences in PO and SM responsibilities

	Product Owner		Scrum Master
Vision	Communicate Vision/Strategy	**Team**	Effective Collaboration
	Value Proposition		Realistic Commitments
	Business Goals, Product Release and Goals		Productive Work Environment
Product	Maintain product Backlog	**Organization**	Educate the Stakeholders
	Direction and Prioritization		Resolve Conflicts
	Collect Feedbacks Available		Organizational Changes required by Scrum
Collaboration	Buy into the process	**Product Owner**	Helps with tools and techniques
	Manage the Stakeholders		Support in decision making and empowerment issues
	Be available		Helps establish Agile practices in Enterprise.

[https://www.c-sharpcorner.com/article/product-owner-role-and-responsibilities/]

Tampereen yliopisto
Tampere University

PO and Product Manager roles

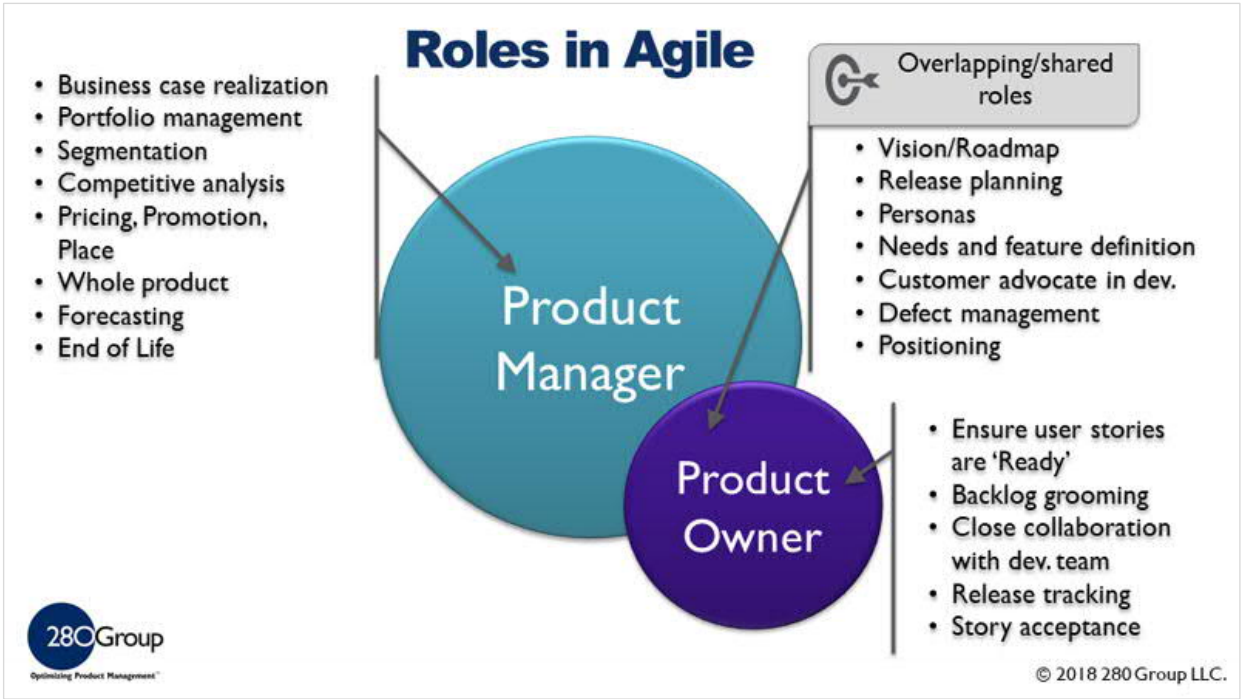
Product Owner	Product Manager
Technology Facing	Customer facing
Identifies Product Needs	Identifies Market Needs
Owns Implementation Process	Owns Product Roadmap
Manages the Dev Team and Works on Daily Scrum Tasks	Manages the Budget and Secures Funding for the Team

[https://www.c-sharpcorner.com/article/product-owner-role-and-responsibilities/]

TABLE 6.1 Product Owner Dos and Don'ts

Do	Don't
Say *what* needs to get done.	Say *how* to do it or *how much* it will take.
Challenge the team.	Bully the team.
Get interested in building a high-performance team.	Focus on the short-term deliveries only.
Practice business-value-driven thinking.	Stick to the original scope and approach "no matter what."
Protect the team from outside noise.	Worry the team with changes that might happen, until they become real.
Incorporate change between sprints.	Allow change to creep into sprints.

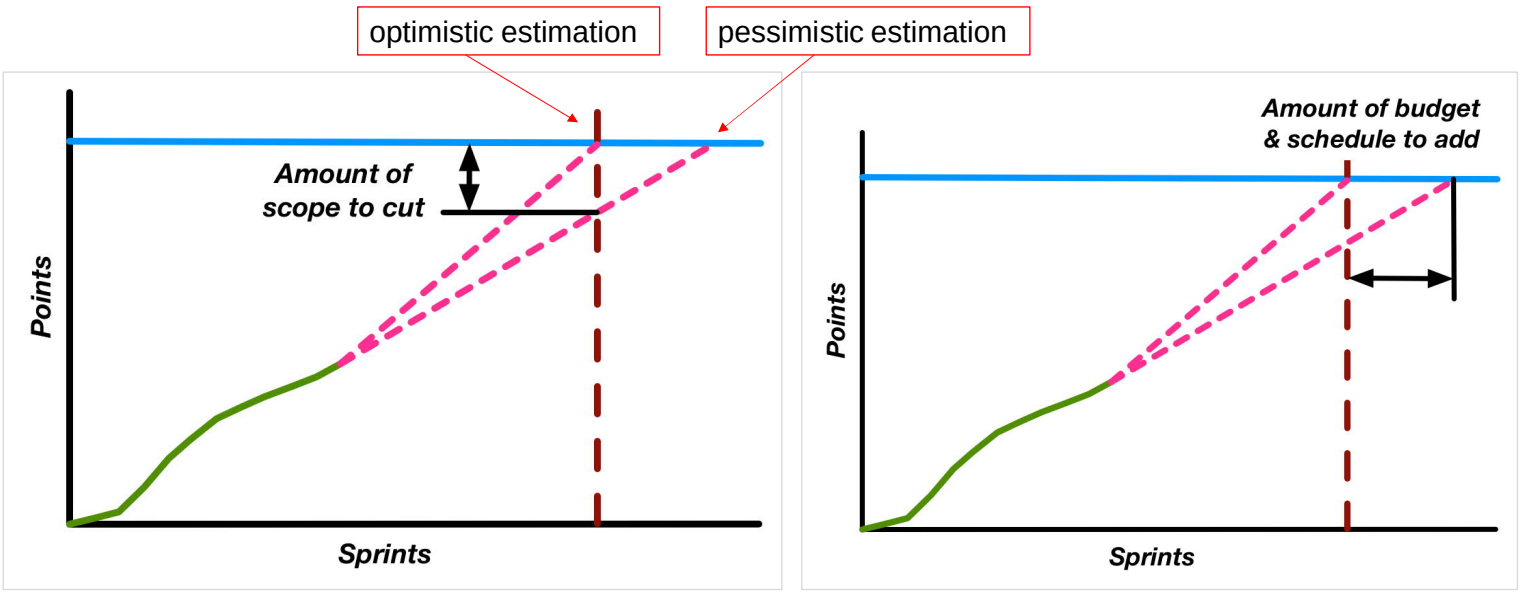
[Agile product management, 2010]



Burndown / burnup charts

Tampereen yliopisto
Tampere University

Burnup chart example, 2



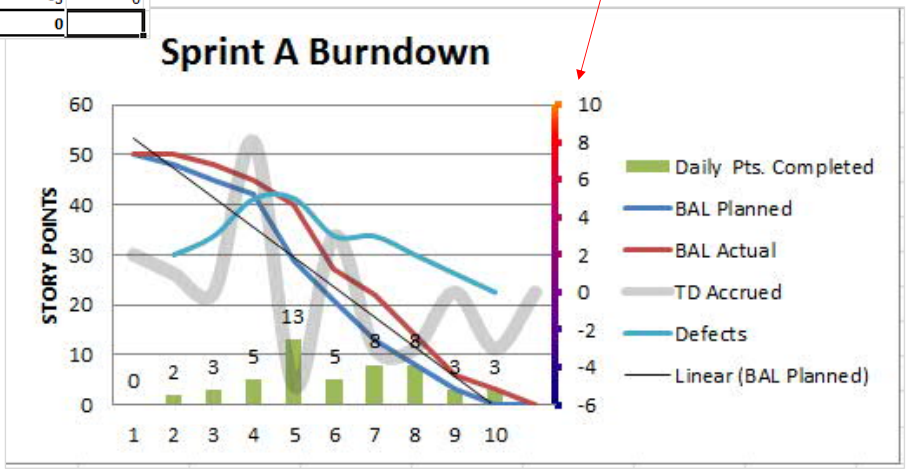
At the end of Scrum projects, there are usually some (less important) work items left in the PB.

Burndown chart, 2

in table and chart format

Day	BD Planned	BD Actual	BAL Planned	BAL Actual	Daily Pts. Completed	TD Accrued	Defects
			50	50			
1	2	0	48	50	0	2	
2	3	2	45	48	2	1	2
3	3	3	42	45	3	0	3
4	13	5	29	40	5	8	5
5	8	13	21	27	13	-5	5
6	8	5	13	22	5	3	3
7	5	8	8	14	8	-3	3
8	5	8	3	6	8	-3	2
9	3	3	0	3	3	0	1
10	0	3	0	0	3	-3	0
	50	50				0	

TD count



BAL = burndown all left
BD = burndown (daily points)
TD = technical debt

[https://qualitasrebus.com/]

Tampereen yliopisto
Tampere University

Scrum-BUT

Lean

#93

Get More Refcardz! Visit refcardz.com

DZone Refcardz

brought to you by... VERSIONONE Simplifying Software Delivery

CONTENTS INCLUDE:

- About Lean Software Development
- Getting Started
- Zero Practices
- Daily Standup
- Automated Testing
- Continuous Integration and more...

Getting Started with Lean Software Development

By Curt Hibbs, Steve Jewett, and Mike Sullivan

ABOUT LEAN SOFTWARE DEVELOPMENT

Lean Software Development is an outgrowth of the larger Lean movement that includes areas such as manufacturing, supply chain management, product development, and back-office operations. Its goal is the same: deliver value to the customer more quickly by eliminating waste and improving quality. Though software development differs from the manufacturing context in which Lean was born, it draws on many of the same principles.

Seven Principles of Lean Software Development

Lean Software Development embodies seven principles, originally described in the book Implementing Lean Software Development: From Concept to Cash[1], by Mary and Tom Poppendieck. Each of these seven principles contributes to the "leaning out" of a software development process.

to accomplish a task, recognizing their efforts, and standing by them when those efforts are unsuccessful.

Optimize the Whole

Optimizing the whole development process generates better results than optimizing local processes in isolation, which is usually done the expense of other local processes.

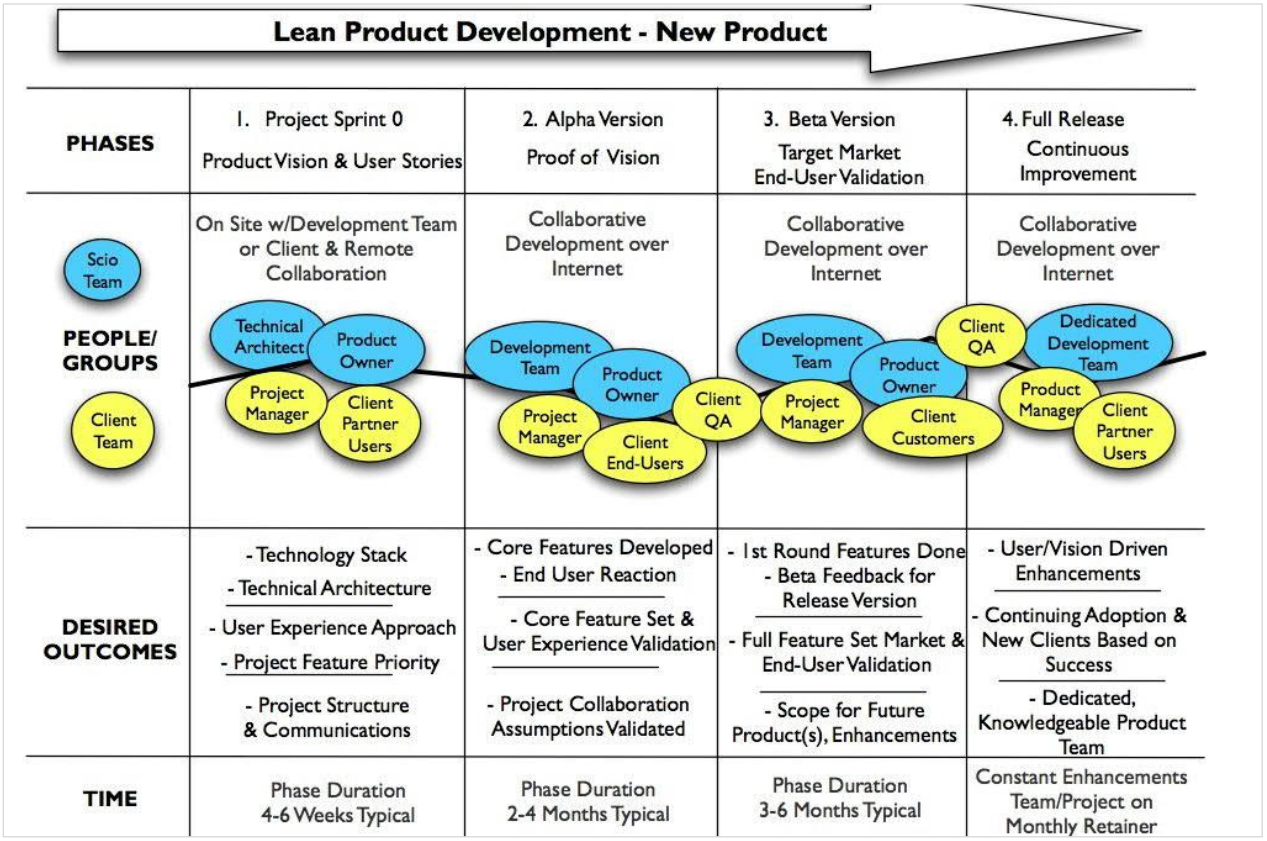
Lean vs. Agile

Comparing Lean and Agile software development reveals they share many characteristics, including the quick delivery of value to the customer, but they differ in two significant ways: scope and focus. The narrow scope of Agile addresses the development of software and focuses on adaptability to deliver quickly. Lean looks at the bigger picture, the context in which development occurs, and delivers value quickly by focusing on the elimination of waste. As it turns out, they are complementary, and real world processes often draw from both.

Tampereen yliopisto
Tampere University

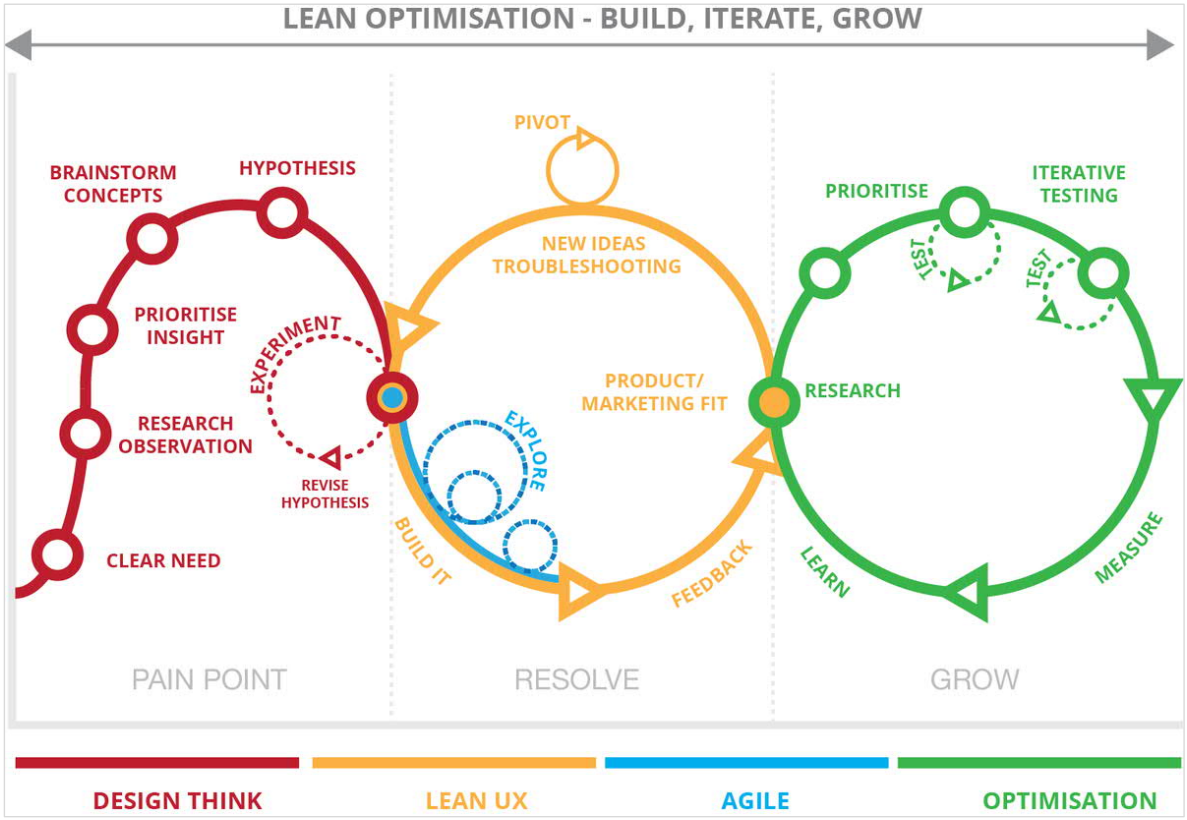
One example

Scio = consulting company



[https://sciodev.com/blog/lean-software-product-development/]

Tampereen yliopisto
Tampere University

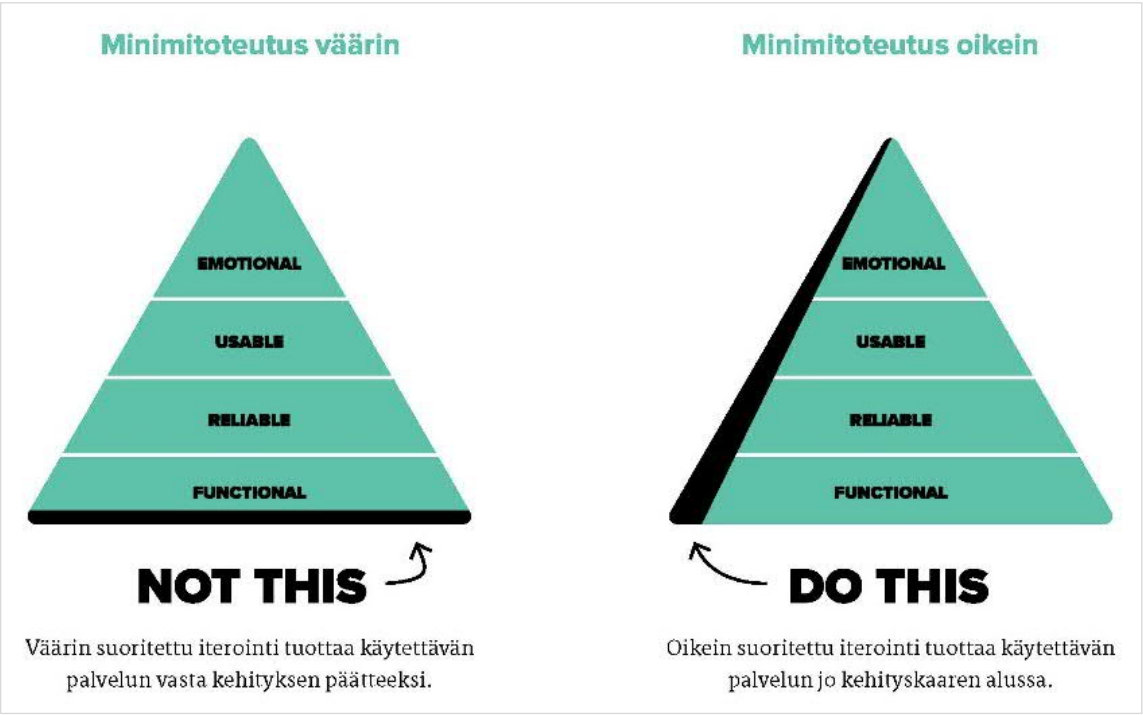


[Craig Sullivan, 2015]

Minimum viable product (MVP)

MVP

[Solinor, 2016]

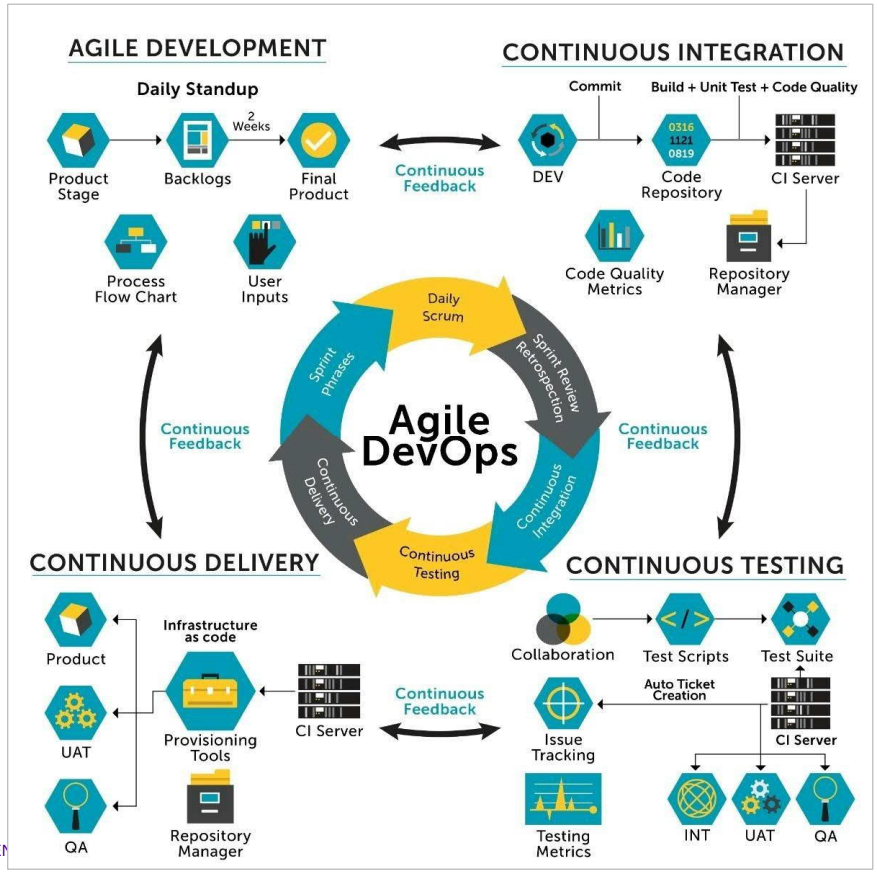


Tampereen yliopisto
Tampere University

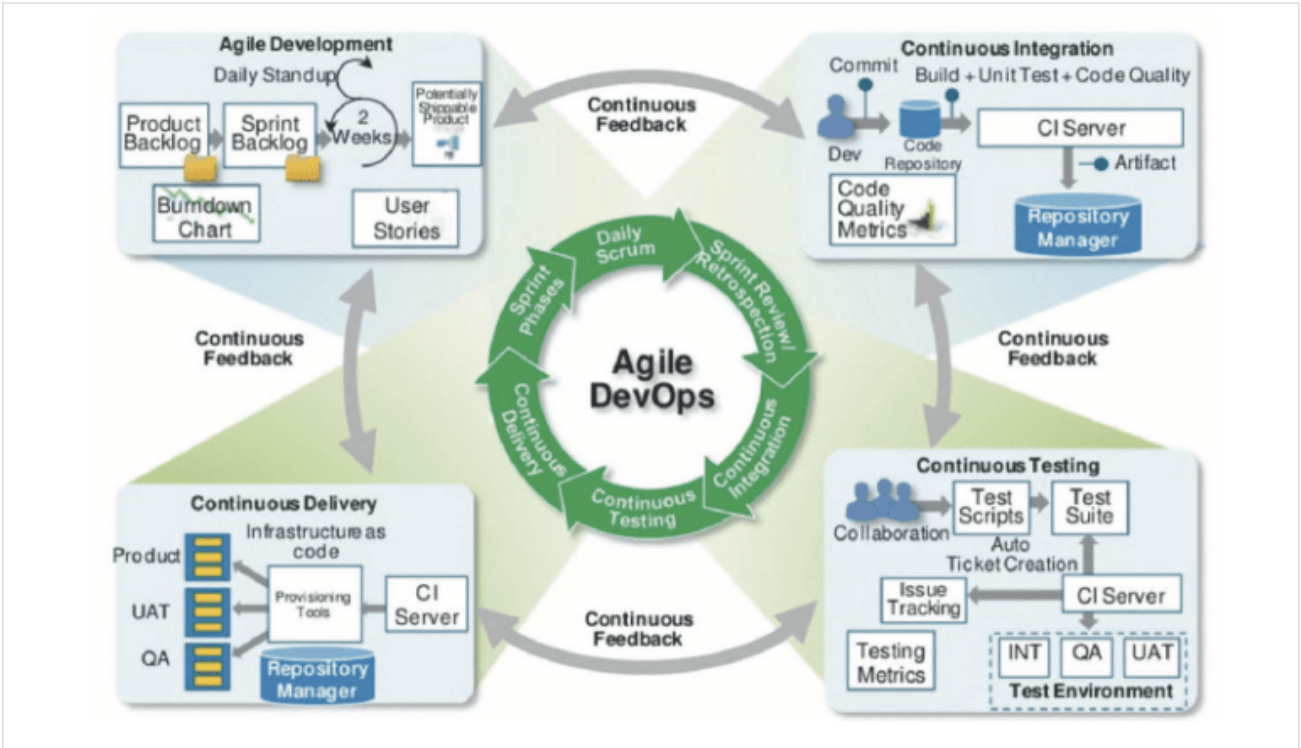
Agile model driven development (AMDD)

Tampereen yliopisto
Tampere University

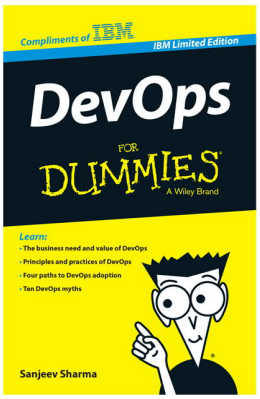
DevOps



[https://pbs.twimg.com/media/DcI-rq-VQAA6tsG.jpg]



[https://www.sjtechcorp.com/casestudies/devops-at-the-u-s-treasury/]



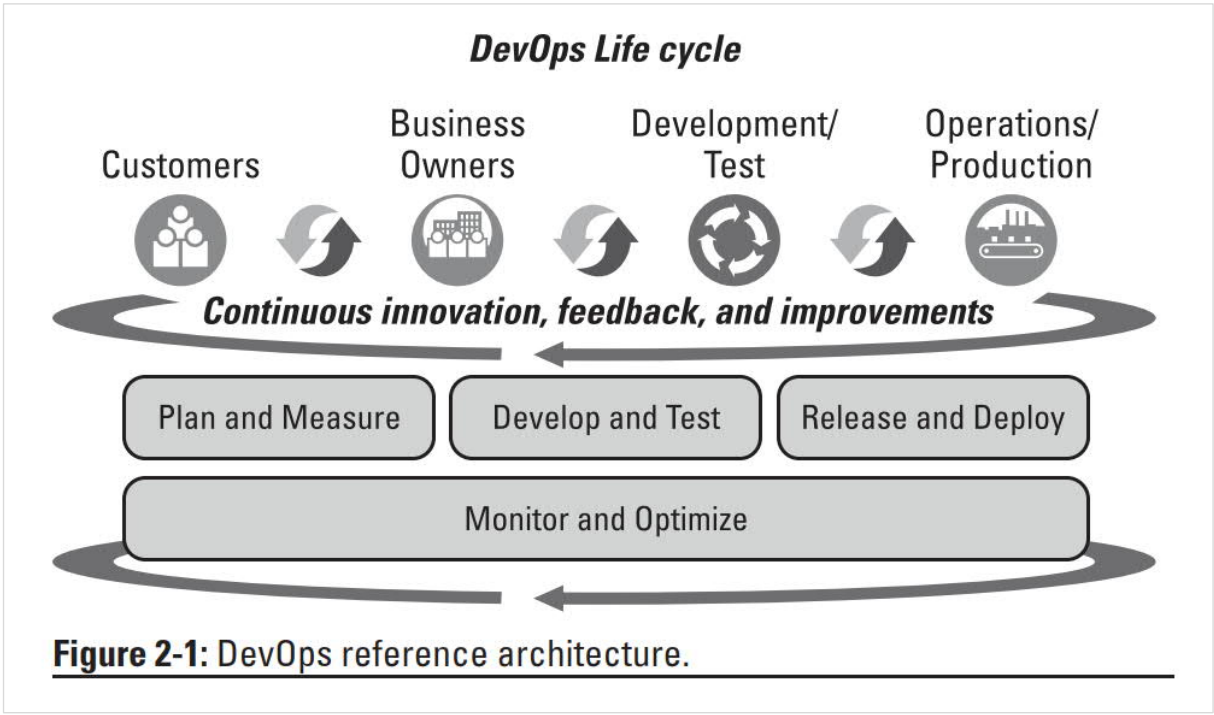
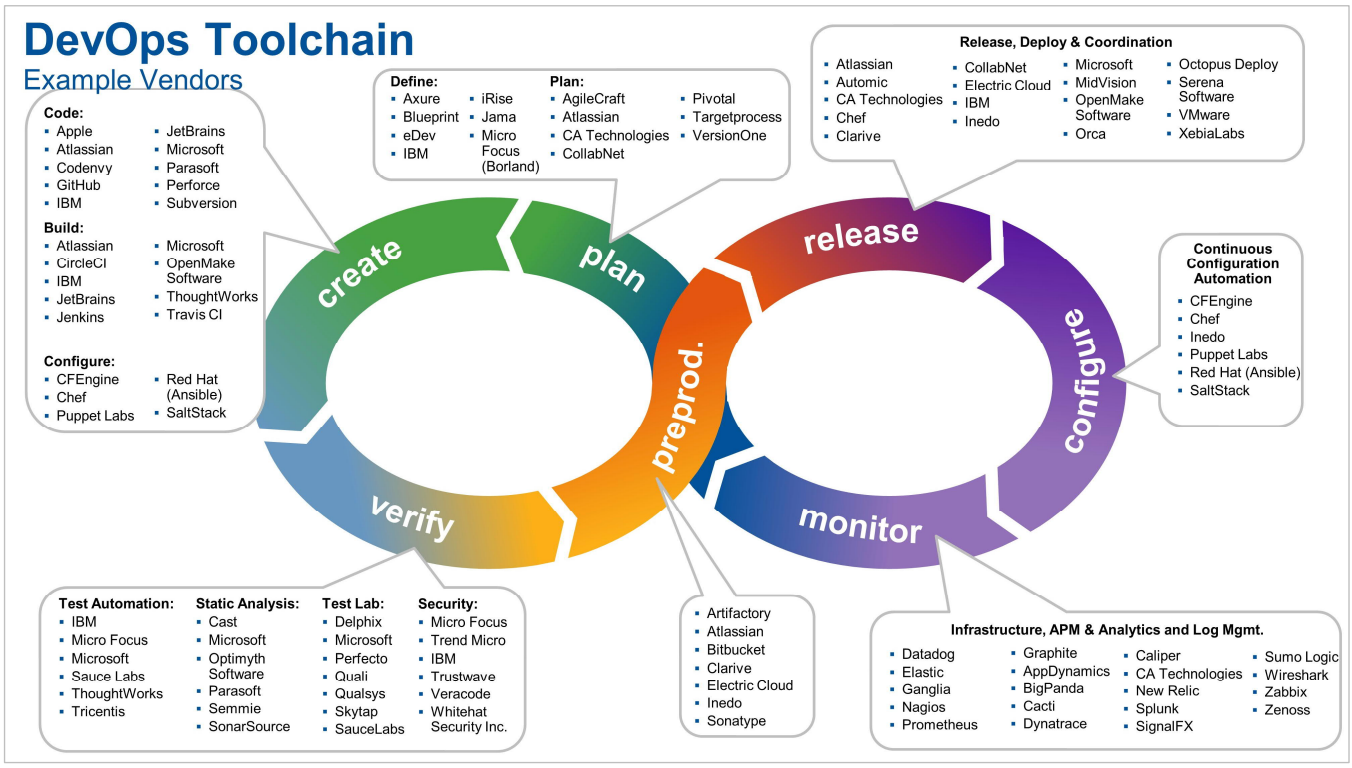


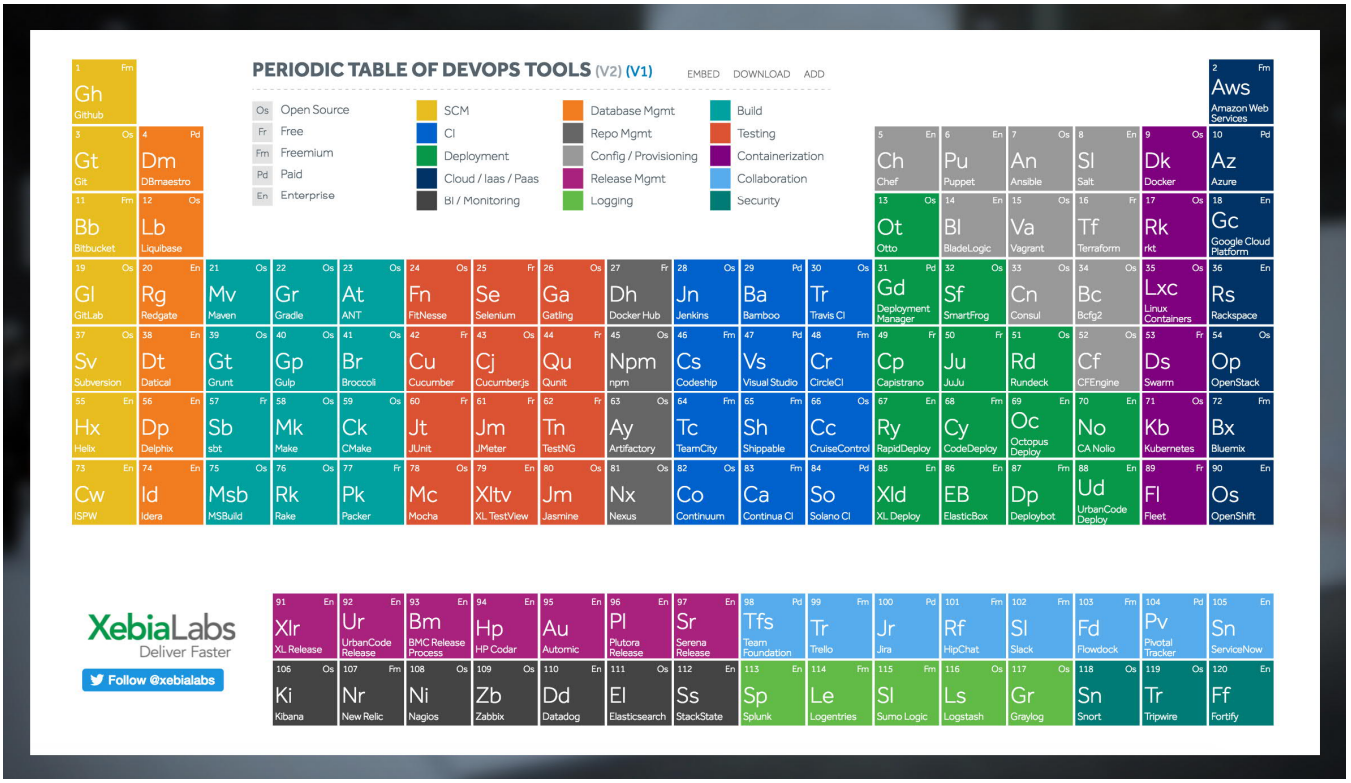
Figure 2-1: DevOps reference architecture.



[https://blogs.gartner.com/christopher-little/2019/01/09/jan-2019-devops-agenda/]

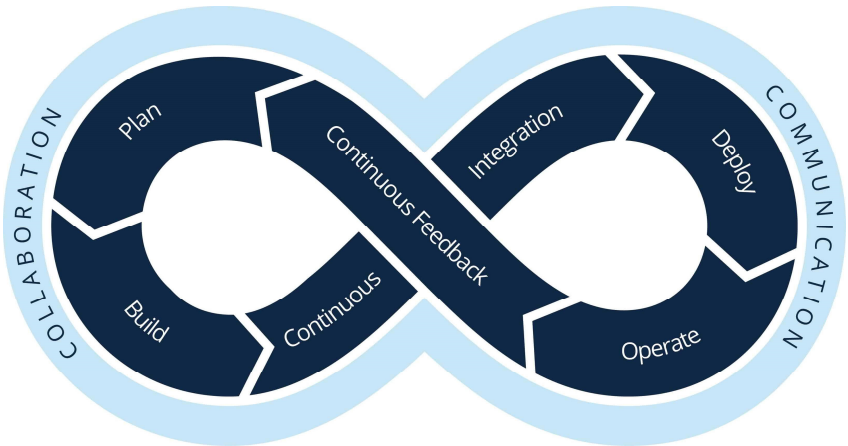
Tampereen yliopisto
Tampere University

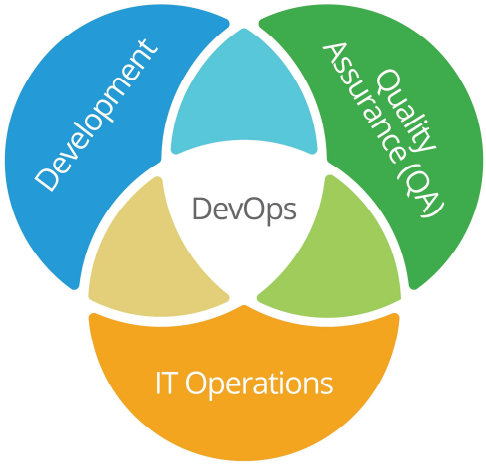
A lot of tools available



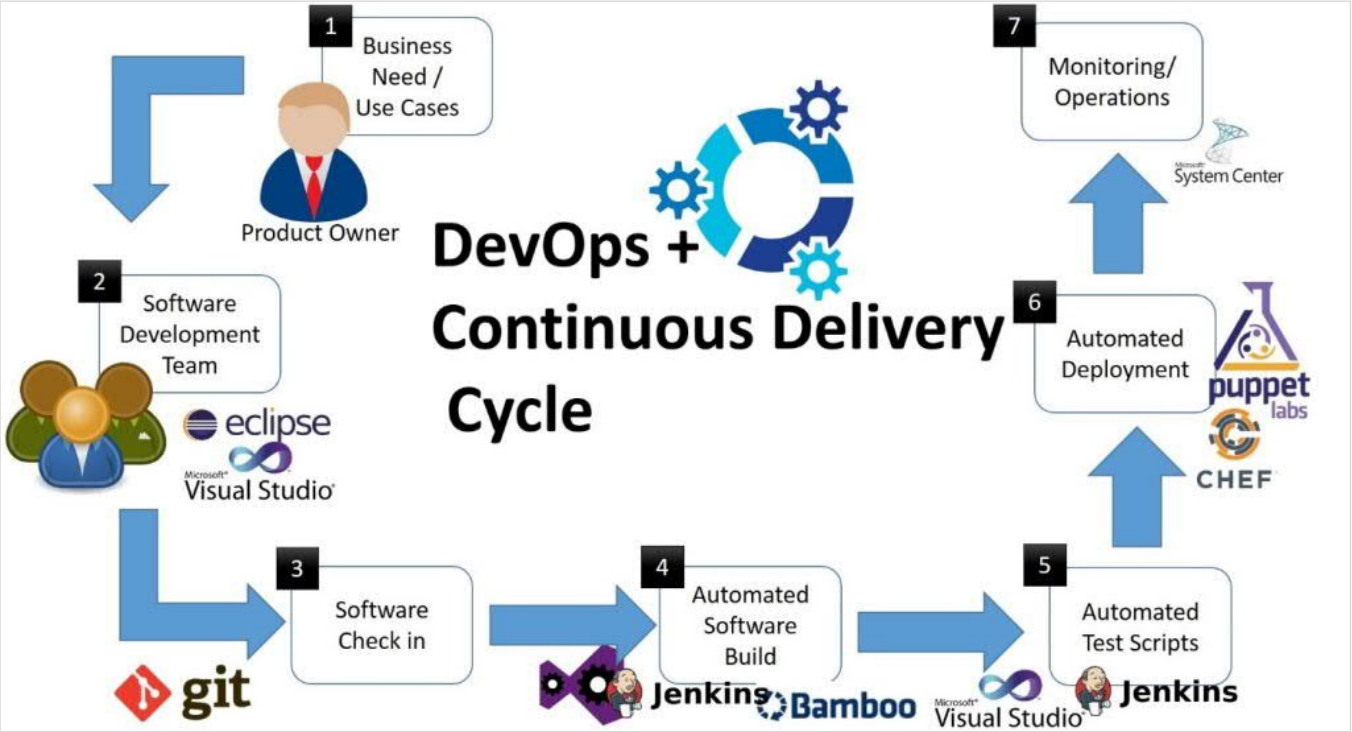
Tampereen yliopisto
Tampere University

DevOps is an organizational culture that leverages core principles and technical practices (for example continuous delivery) to optimize application performance and infrastructure and deliver value as a strategic asset.
The DevOps movement is an effort to transform the way various stakeholders interact, communicate, and work together in the software development lifecycle.





[www.smartsheet.com/devops]



[http://digitalcto.com/can-you-build-software-faster-cheaper-and-better/]

Some daily software deployment numbers

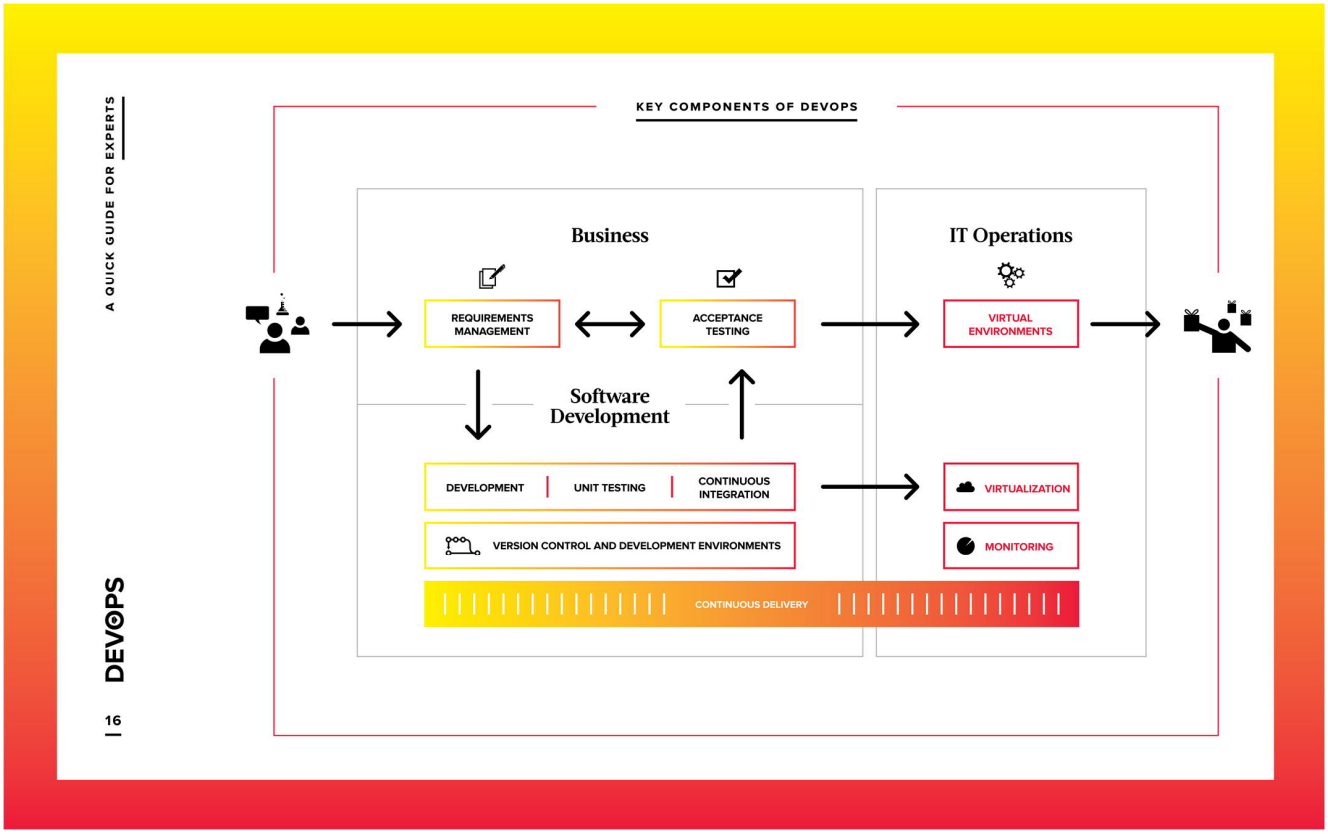
infrastructure engineering, called *The Phoenix Project*. The book reads like a novel while it explains DevOps principles.

A table at the back of the book shows how often different companies deploy to the release environment:

Company	Deployment Frequency
Amazon	23,000 per day
Google	5,500 per day
Netflix	500 per day
Facebook	1 per day
Twitter	3 per week
Typical enterprise	1 every 9 months

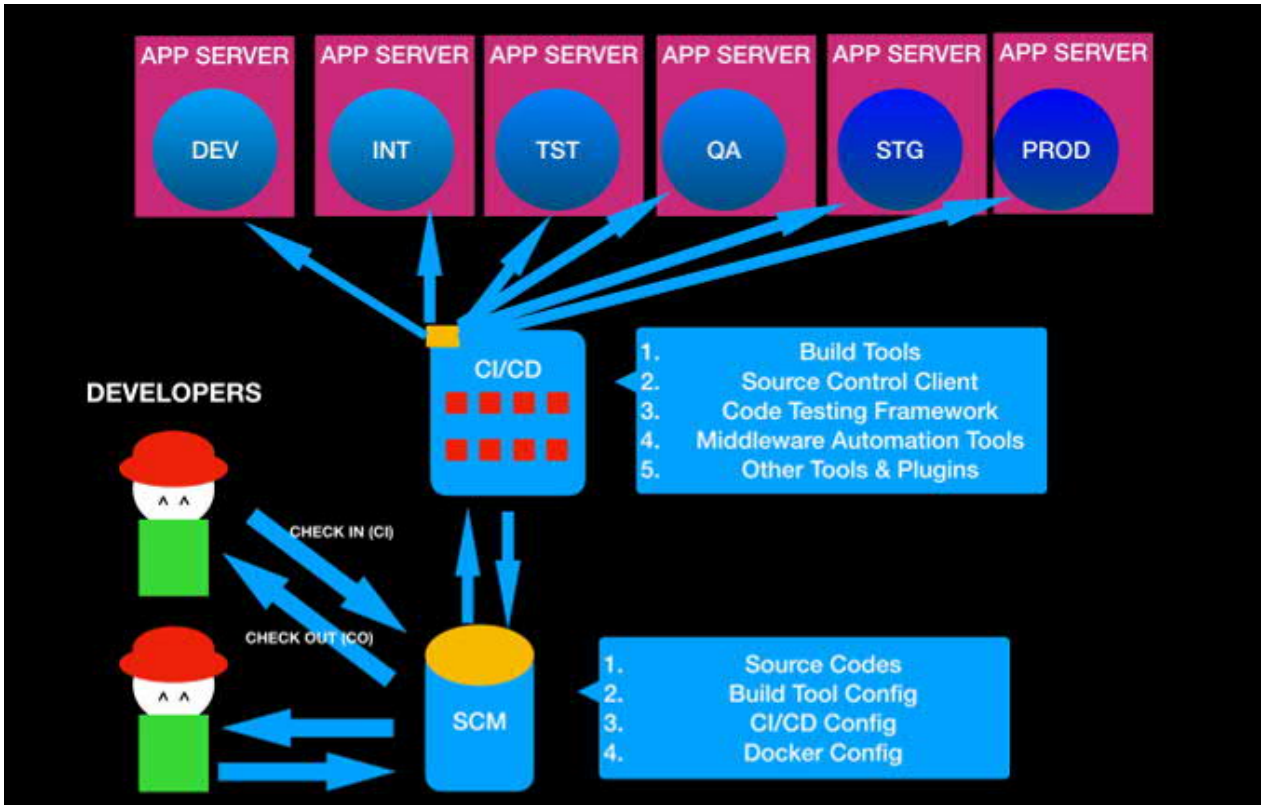
How are the frequency rates of Amazon, Google, and Netflix even possible? It's because these companies have figured out how to make a nearly perfect DevOps pipeline.

[opensource.com/article/19/4/devops-pipeline]



[Eficode DevOps guide]

DevOps pipeline

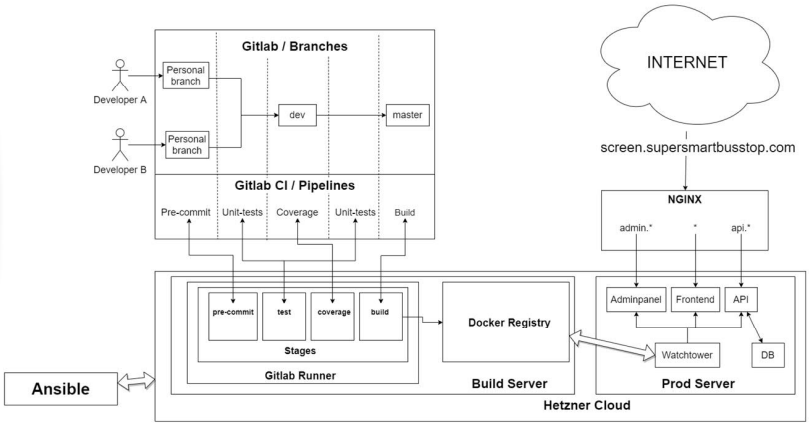


[opensource.com/article/19/4/devops-pipeline]

Server infrastructure & CI/CD

- 121 merge requests
- 540 pipelines
- 798 jobs

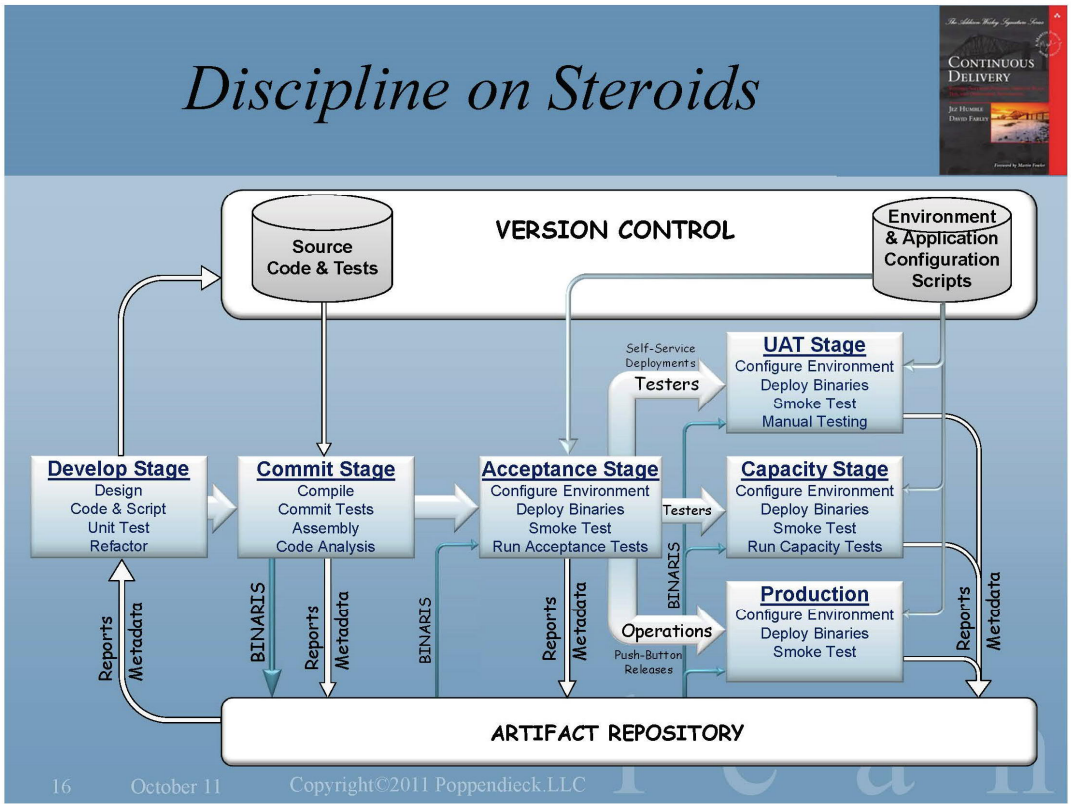
TIE-PROJ 2019-20,
G05,
Smart bus stop



CI = continuous integration

CD = continuous delivery

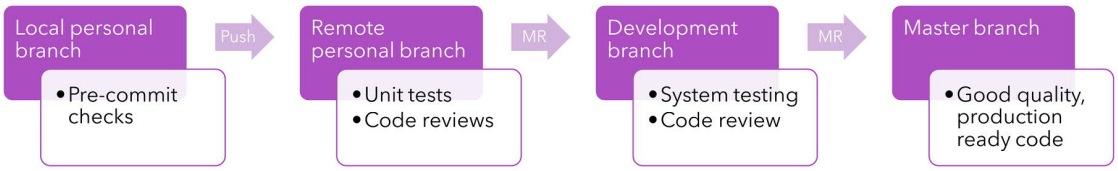
UAT = user acceptance testing



Tampereen yliopisto
Tampere University

Version control & QA

TIE-PROJ 2019-20, G05, GitLab use, Smart bus stop



*MR = Merge Request

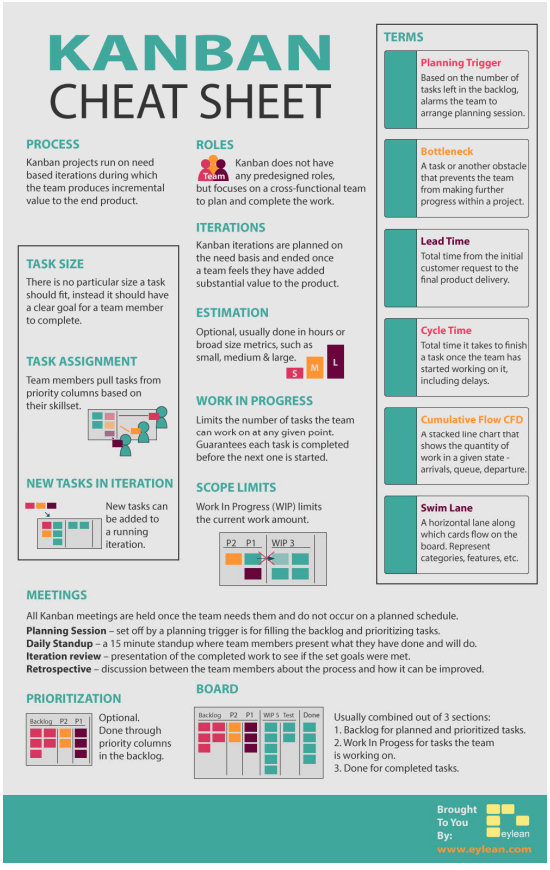
Tampereen yliopisto
Tampere University

Scaled agile framework (SAFe)

Kanban

Tampereen yliopisto
Tampere University

kanban



KANBAN
CHEAT SHEET

PROCESS
Kanban projects run on need based iterations during which the team produces incremental value to the end product.

ROLES
Team
Kanban does not have any predesigned roles, but focuses on a cross-functional team to plan and complete the work.

ITERATIONS
Kanban iterations are planned on the need basis and ended once a team feels they have added substantial value to the product.

ESTIMATION
Optional, usually done in hours or broad size metrics, such as small, medium & large.
S M L

TASK SIZE
There is no particular size a task should fit, instead it should have a clear goal for a team member to complete.

TASK ASSIGNMENT
Team members pull tasks from priority columns based on their skillset.

WORK IN PROGRESS
Limits the number of tasks the team can work on at any given point. Guarantees each task is completed before the next one is started.

NEW TASKS IN ITERATION
New tasks can be added to a running iteration.

SCOPE LIMITS
Work In Progress (WIP) limits the current work amount.
P2 P1 WIP 3

TERMS

Planning Trigger
Based on the number of tasks left in the backlog, alarms the team to arrange planning session.

Bottleneck
A task or another obstacle that prevents the team from making further progress within a project.

Lead Time
Total time from the initial customer request to the final product delivery.

Cycle Time
Total time it takes to finish a task once the team has started working on it, including delays.

Cumulative Flow CFD
A stacked line chart that shows the quantity of work in a given state - arrivals, queue, departure.

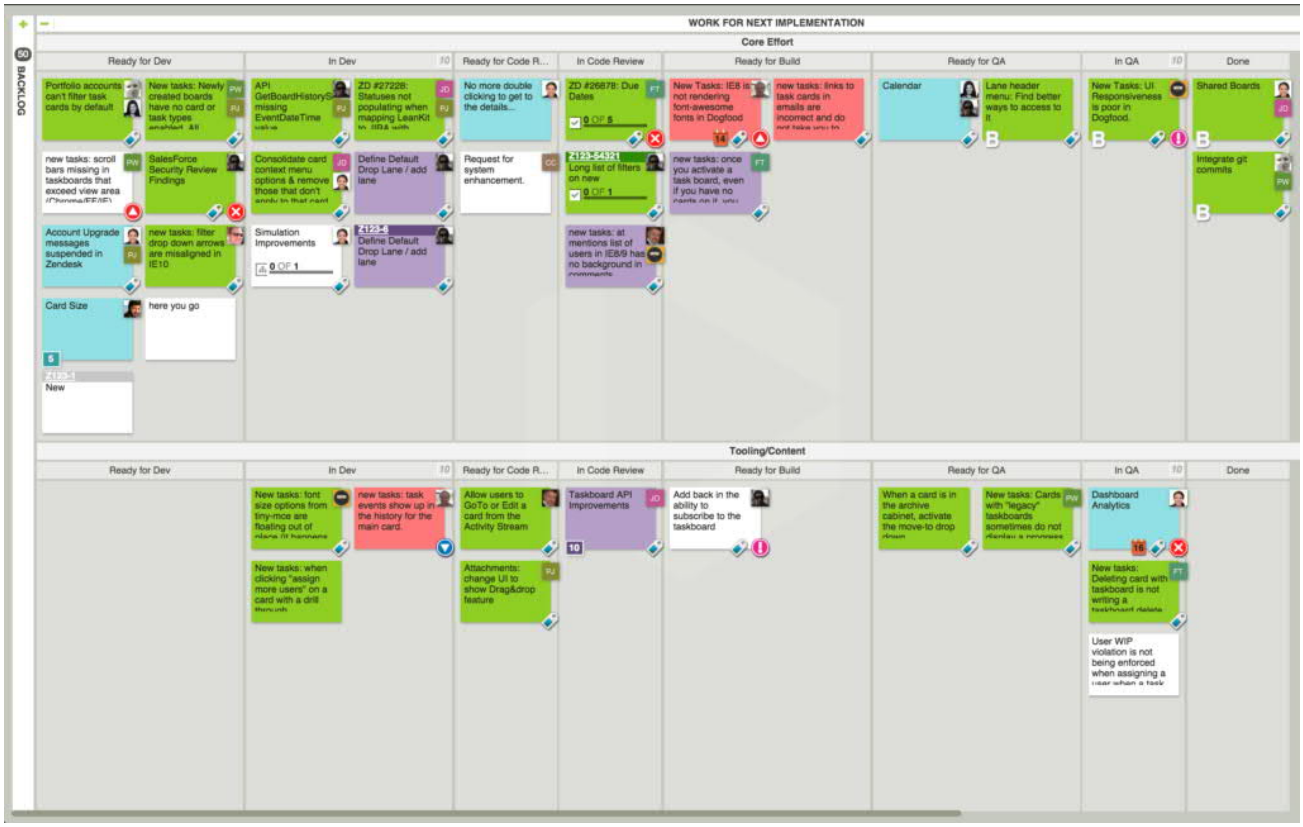
Swim Lane
A horizontal lane along which cards flow on the board. Represent categories, features, etc.

MEETINGS
All Kanban meetings are held once the team needs them and do not occur on a planned schedule.
Planning Session – set off by a planning trigger is for filling the backlog and prioritizing tasks.
Daily Standup – a 15 minute standup where team members present what they have done and will do.
Iteration review – presentation of the completed work to see if the set goals were met.
Retrospective – discussion between the team members about the process and how it can be improved.

PRIORITIZATION
Backlog P2 P1
Optional. Done through priority columns in the backlog.

BOARD
Backlog P2 P1 WIP 5 Test Done
Usually combined out of 3 sections:
1. Backlog for planned and prioritized tasks.
2. Work In Progess for tasks the team is working on.
3. Done for completed tasks.

Brought To You By: eylean
www.eylean.com



[www.leankit.com]

Tampereen yliopisto
Tampere University

Projects in controlled environments (PRINCE)

Scrumban

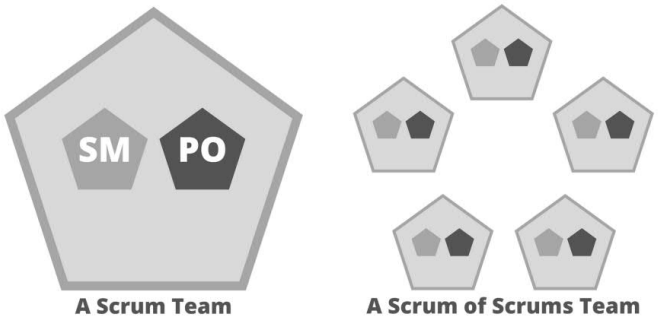
DIFFERENCE BETWEEN	
SCRUM	**KANBAN**
Timeboxed iterations prescribed.	Timeboxed iterations optional. Can have separate cadences for planning, release, and process improvement. Can be event driven instead of timeboxed.
Team commits to a specific amount of work for this iteration.	Commitment optional.
Uses Velocity as default metric for planning and process improvement.	Uses Lead time as default metric for planning and process improvement.
Cross-functional teams prescribed.	Cross-functional teams optional. Specialist teams allowed.
Items must be broken down so they can be completed within 1 sprint.	No particular item size is prescribed.
Burndown chart prescribed.	No particular type of diagram is prescribed.
WIP limited indirectly (per sprint).	WIP limited directly (per workflow state).
Estimation prescribed.	Estimation optional.
Can not add items to ongoing iteration.	Can add new items whenever capacity is available.
A sprint backlog is owned by one specific team.	A Kanban board may be shared by multiple teams or individuals.
Prescribes 3 roles (PO/SM/Team).	Doesn't prescribe any roles.
A Scrum board is reset between each sprint.	A Kanban board is persistent.
Prescribes a prioritized product backlog.	Prioritization is optional.

[www.cuelogic.com]

Scrum of scrums

Scrum of Scrums, 1

A Scrum of Scrums operates as if it were a Scrum Team, satisfying the Team Process component with scaled versions of the Scrum roles, events, and artifacts. While the Scrum Guide defines the optimal team size as being 3 to 9 people, Harvard research has determined that optimal team size is 4.6 people (on average), therefore the optimal number of teams in an Scrum of Scrums is 4 or 5.



NOTE: In the above and following diagrams, light-grey outlined pentagons represent a team. Where applicable, we have chosen to represent the SM & PO roles as smaller pentagons. These diagrams are meant to be examples only, as each organizational diagram may differ greatly.

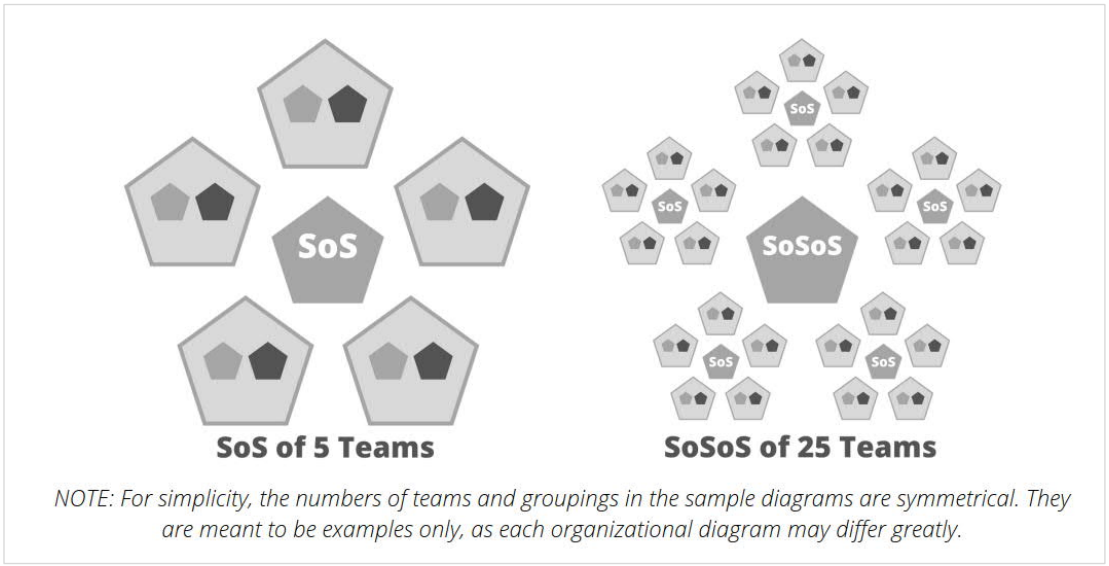
As a dynamic group, the teams composing the Scrum of Scrums are responsible for a fully integrated set of potentially shippable increments of product at the end of every Sprint.

Optimally, they carry out all of the functions required to release value directly to customers.

Tampereen yliopisto
Tampere University

Scrum of Scrums, 2

Depending upon the size of an implementation, more than one Scrum of Scrums may be needed to deliver a very complex product. In such cases, a Scrum of Scrum of Scrums (SoSoS) can be created out of multiple Scrums of Scrums. Each of these will have scaled versions of each Scrum of Scrums' roles, artifacts, and events.



Tampereen yliopisto
Tampere University